

TRichView Components

TRichView © trichview.com

Table of Contents

Part I TRichView Reference 26

1..Overview 28

Parts of Documents 29
TRichView Item Types 29
"Checkpoints" 30
Items' "Tags" 34
Hypertext Overview 35
Paragraphs 37
Borders and Background of Paragraphs 39
Styles and StyleTemplates 40
Hidden Text 42

Basic Operations 43
Building a Document 43
Scrolling 46
Selecting 47
Clipboard 48
Searching and Replacing 49
Obtaining Items 50
Modifying Items 51

Editing 52
RichView Edit: Inserting Items in Position of Caret 52
Undo and Redo 54
Drag and Drop 55

Graphics 56
Animated Images 56
Smooth Image Scaling 57
Semitransparent Objects 57

Saving and Loading 58
Saving and Loading 58
RVF (RichView Format) 59
Export to HTML 62

Internationalization 64
Unicode in TRichView 64
Bidirectional text 69

Spelling Check 69
Live spelling check 69

Important Technical Information 71
Viewer vs Editor 71
Controls - Documents - Items 72
Valid Documents 74
Units of Measurement 74
Custom drawing 76
Cursors 77
RVF Specification 77

2..Overview of Item Types 85

Overview 86
Text Items 88
Tabulators 89
Pictures 90
Hot-Pictures 93

Breaks ... 93
Controls ... 94
Bullets ... 96
Hotspots ... 97
Tables ... 98
List Markers ... 99
Labels ... 101
Numbered Sequences ... 102
Endnotes ... 103
Footnotes ... 104
Sidenotes ... 105
Text Boxes ... 107
References to Notes ... 108
Page Numbers ... 109
Page Counts ... 110
Custom Item Types ... 111
3 Overview of Tables ... 111
Tables in RichView ... 111
Addressing Items in Table Cells ... 112
Table Cells ... 113
Colors and Layout of Tables ... 114
Selection in Table ... 116
Editing Cells ... 117
Cell Merging ... 118
Table Operations ... 119
Undo/redo in Tables ... 121
Export of Tables ... 123
Table Resizing ... 124
Example: table to text ... 125
Example: dividing table ... 126
4 Components ... 127
Visual Components ... 128
TRichView ... 128
TRichView : Component editor ... 134
Properties ... 135
TRichView .Allow Selection ... 135
TRichView .AnimationMode ... 135
TRichView .BackgroundBitmap ... 136
TRichView .BackgroundStyle ... 136
TRichView .BiDiMode ... 137
TRichView .BottomMargin ... 137
TRichView .ClearLeft ... 138
TRichView .ClearRight ... 138
TRichView .Color ... 139
TRichView .CPEventKind ... 139
TRichView .Cursor ... 140
TRichView .Delimiters ... 140
TRichView .DocParameters ... 140
TRichView .DocProperties ... 141
TRichView .DocumentHeight ... 142
TRichView .DoInPaletteMode ... 142
TRichView .FirstItemVisible ... 144
TRichView .FirstJumpNo ... 144
TRichView .ItemCount ... 144
TRichView .LastItemVisible ... 144
TRichView .LeftMargin ... 145
TRichView .LineCount ... 145

TRichView .MaxLength ... 145
TRichView .MaxTextWidth ... 146
TRichView .MinTextWidth ... 146
TRichView .NoteText ... 146
TRichView .Options ... 146
TRichView .PageBreaksBeforeItems ... 149
TRichView .RightMargin ... 149
TRichView .RTFOptions ... 150
TRichView .RTFReadProperties ... 151
TRichView .RVData ... 151
TRichView .RVFOptions ... 152
TRichView .RVFParaStylesReadMode ... 154
TRichView .RVFTextStylesReadMode ... 154
TRichView .RVFWarnings ... 155
TRichView .SelectionHandlesVisible ... 156
TRichView .SingleClick ... 156
TRichView .Style ... 156
TRichView .StyleTemplateInsertMode ... 156
TRichView .TabNavigation ... 157
TRichView .TopMargin ... 158
TRichView .UseStyleTemplates ... 158
TRichView .VAlign ... 159
TRichView .VSmallStep ... 159
TRichView .WordWrap ... 159
Methods ... 160
TRichView .AddBreakExTag and Others ... 160
TRichView .AddBulletExTag and Others ... 160
TRichView .AddCenterLine, AddFromNew Line ... 161
TRichView .AddControlExTag and Others ... 162
TRichView .AddFmt ... 163
TRichView .AddHotPicture and AddHotPictureTag ... 163
TRichView .AddHotspotExTag and Others ... 164
TRichView .AddItem ... 166
TRichView .AddNamedCheckpointExTag and Others ... 166
TRichView .AddNLTag -A- -W- ... 167
TRichView .AddPictureExTag and Others ... 169
TRichView .AddTab ... 170
TRichView .AddText, AddTextFromNew Line (deprecated) ... 171
TRichView .AddTextNL -A -W ... 171
TRichView .AppendFrom ... 172
TRichView .AppendRVFFromStream ... 173
TRichView .AssignSoftPageBreaks ... 174
TRichView .BeginOleDrag ... 174
TRichView .Clear ... 175
TRichView .ClearLiveSpellingResults ... 175
TRichView .ClearSoftPageBreaks ... 176
TRichView .ClientToDocument, DocumentToClient ... 176
TRichView .ConvertDocTo ... 176
TRichView .Copy ... 177
TRichView .CopyDef ... 177
TRichView .CopyImage ... 177
TRichView .CopyRTF ... 178
TRichView .CopyRVF ... 178
TRichView .CopyText -W ... 178
TRichView .Create ... 179
TRichView .DeleteItems ... 179
TRichView .DeleteMarkedStyles ... 180
TRichView .DeleteParas ... 180

TRichView.DeleteSection ... 180
TRichView.DeleteUnusedStyles ... 181
TRichView.Deselect ... 181
TRichView.Destroy ... 181
TRichView.FindCheckpointByName ... 182
TRichView.FindCheckpointByTag ... 182
TRichView.FindControlItemNo ... 182
TRichView.Format ... 182
TRichView.FormatTail ... 183
TRichView.GetBreakInfo ... 183
TRichView.GetBulletInfo ... 184
TRichView.GetCheckpointByNo ... 185
TRichView.GetCheckpointInfo ... 185
TRichView.GetCheckpointItemNo ... 186
TRichView.GetCheckpointNo ... 186
TRichView.GetCheckpointXY ... 186
TRichView.GetCheckpointY ... 186
TRichView.GetCheckpointYEx ... 187
TRichView.GetControlInfo ... 187
TRichView.GetFirstCheckpoint ... 188
TRichView.GetFocusedItem ... 188
TRichView.GetHotspotInfo ... 189
TRichView.GetItem ... 190
TRichView.GetItemAt ... 190
TRichView.GetItemCheckpoint ... 191
TRichView.GetItemCoords ... 191
TRichView.GetItemCoordsEx ... 192
TRichView.GetItemExtraIntProperty -Ex ... 193
TRichView.GetItemExtraStrProperty -Ex ... 193
TRichView.GetItemNo ... 194
TRichView.GetItemPara ... 194
TRichView.GetItemStyle ... 194
TRichView.GetItemTag ... 195
TRichView.GetItemText -A -W ... 195
TRichView.GetItemVAlign ... 196
TRichView.GetJumpPointItemNo ... 196
TRichView.GetJumpPointLocation ... 197
TRichView.GetJumpPointY ... 197
TRichView.GetLastCheckpoint ... 198
TRichView.GetLineNo ... 198
TRichView.GetListMarkerInfo ... 199
TRichView.GetNextCheckpoint ... 199
TRichView.GetOffsAfterItem ... 200
TRichView.GetOffsBeforeItem ... 201
TRichView.GetPictureInfo ... 201
TRichView.GetPrevCheckpoint ... 202
TRichView.GetSelectedImage ... 202
TRichView.GetSelectionBounds ... 203
TRichView.GetSelText ... 204
TRichView.GetTextInfo ... 204
TRichView.GetWordAt -A -W ... 205
TRichView.InsertRVFFromStream ... 206
TRichView.IsFromNewLine ... 207
TRichView.IsParaStart ... 208
TRichView.LiveSpellingValidateWord ... 208
TRichView.LoadFromStream ... 208
TRichView.LoadRTF ... 209
TRichView.LoadRTFFromStream ... 210

TRichView.LoadRVF ... 211
TRichView.LoadRVFFromStream ... 211
TRichView.LoadText -W ... 212
TRichView.LoadTextFromStream -W ... 213
TRichView.MarkStylesInUse ... 213
TRichView.Reformat ... 214
TRichView.RefreshListMarkers ... 214
TRichView.RefreshListSequences ... 214
TRichView.RemoveCheckpoint ... 214
TRichView.ResetAnimation ... 215
TRichView.SaveDocX ... 215
TRichView.SaveDocXToStream ... 216
TRichView.SaveHTML ... 216
TRichView.SaveHTMLEx ... 217
TRichView.SaveHTMLToStream ... 218
TRichView.SaveHTMLToStreamEx ... 218
TRichView.SavePicture ... 219
TRichView.SaveRTF ... 219
TRichView.SaveRTFToStream ... 220
TRichView.SaveRVF ... 220
TRichView.SaveRVFToStream ... 220
TRichView.SaveText -W ... 221
TRichView.SaveTextToStream -W ... 221
TRichView.SearchText -A -W ... 222
TRichView.SelectAll ... 223
TRichView.SelectControl ... 223
TRichView.SelectionExists ... 224
TRichView.SelectWordAt ... 224
TRichView.SetAddParagraphMode ... 225
TRichView.SetBreakInfo ... 226
TRichView.SetBulletInfo ... 227
TRichView.SetCheckpointInfo ... 227
TRichView.SetControlInfo ... 228
TRichView.SetHotspotInfo ... 229
TRichView.SetItemExtraIntProperty -Ex ... 230
TRichView.SetItemExtraStrProperty -Ex ... 231
TRichView.SetItemTag ... 231
TRichView.SetItemText -A -W ... 232
TRichView.SetItemVAlign ... 233
TRichView.SetListMarkerInfo ... 233
TRichView.SetPictureInfo ... 234
TRichView.SetSelectionBounds ... 235
TRichView.StartAnimation ... 237
TRichView.StartLiveSpelling ... 237
TRichView.StopAnimation ... 237
TRichView.UpdatePaletteInfo ... 238
Events ... 238
TRichView.OnAddStyle ... 238
TRichView.OnCheckpointVisible ... 238
TRichView.OnControlAction ... 239
TRichView.OnCopy ... 240
TRichView.OnGetItemCursor ... 240
TRichView.OnHTMLSaveImage ... 241
TRichView.OnImportPicture ... 242
TRichView.OnItemAction ... 244
TRichView.OnItemHint ... 246
TRichView.OnJump ... 247
TRichView.OnPaint ... 247

TRichView.OnProgress ... 248
TRichView.OnReadHyperlink ... 248
TRichView.OnRVDblClick ... 250
TRichView.OnRVFControlNeeded ... 251
TRichView.OnRVFImageListNeeded ... 252
TRichView.OnRVFPictureNeeded ... 252
TRichView.OnRVMouseDown ... 253
TRichView.OnRVMouseMove ... 254
TRichView.OnRVMouseUp ... 254
TRichView.OnRVRightClick ... 255
TRichView.OnSaveComponentToFile ... 256
TRichView.OnSaveDocXExtra ... 257
TRichView.OnSaveHTMLExtra ... 259
TRichView.OnSaveImage2 ... 260
TRichView.OnSaveItemToFile ... 261
TRichView.OnSaveParaToHTML ... 263
TRichView.OnSaveRTFExtra ... 263
TRichView.OnSelect ... 265
TRichView.OnSpellingCheck ... 265
TRichView.OnStyleTemplatesChange ... 266
TRichView.OnURLNeeded ... 266
TRichView.OnWriteHyperlink ... 267
Classes of Properties ... 268
TRVRTFReaderProperties ... 268
Properties ... 269
TRVRTFReaderProperties.AutoHideTableGridLines ... 269
TRVRTFReaderProperties.BasePathLinks ... 269
TRVRTFReaderProperties.CharsetForUnicode ... 269
TRVRTFReaderProperties.ConvertHighlight ... 270
TRVRTFReaderProperties.ErrorCode ... 270
TRVRTFReaderProperties.ExtractMetafileBitmaps ... 271
TRVRTFReaderProperties.IgnoreBookmarks ... 271
TRVRTFReaderProperties.IgnoreFields ... 271
TRVRTFReaderProperties.IgnoreNotes ... 271
TRVRTFReaderProperties.IgnorePictures ... 271
TRVRTFReaderProperties.IgnoreSequences ... 272
TRVRTFReaderProperties.IgnoreTables ... 272
TRVRTFReaderProperties.LineBreaksAsParagraphs ... 272
TRVRTFReaderProperties.ParaStyleMode ... 272
TRVRTFReaderProperties.ParaStyleNo ... 273
TRVRTFReaderProperties.ReadDocParameters ... 273
TRVRTFReaderProperties.SkipHiddenText ... 273
TRVRTFReaderProperties.StoreImagesFileNames ... 273
TRVRTFReaderProperties.TextStyleMode ... 274
TRVRTFReaderProperties.TextStyleNo ... 274
TRVRTFReaderProperties.UnicodeMode ... 274
TRVRTFReaderProperties.UseCharsetForUnicode ... 275
TRVRTFReaderProperties.UseHypertextStyles ... 275
TRVRTFReaderProperties.UseSingleCellPadding ... 275
Methods ... 276
TRVRTFReaderProperties.SetFooter ... 276
TRVRTFReaderProperties.SetHeader ... 276
TRVDocParameters ... 277
Properties ... 277
TRVDocParameters.Author ... 277
TRVDocParameters.BottomMargin ... 277
TRVDocParameters.Comments ... 278
TRVDocParameters.FacingPages ... 278

TRVDocParameters.FooterY ... 278
TRVDocParameters.HeaderY ... 278
TRVDocParameters.LeftMargin ... 278
TRVDocParameters.MirrorMargins ... 279
TRVDocParameters.Orientation ... 279
TRVDocParameters.PageHeight ... 279
TRVDocParameters.PageWidth ... 279
TRVDocParameters.RightMargin ... 279
TRVDocParameters.Title ... 280
TRVDocParameters.TitlePage ... 280
TRVDocParameters.TopMargin ... 280
TRVDocParameters.Units ... 280
TRVDocParameters.ZoomMode ... 281
TRVDocParameters.ZoomPercent ... 281
Methods ... 281
TRVDocParameters.ConvertToUnits ... 281
TRVDocParameters.Reset ... 281
TRVDocParameters.ResetLayout ... 281
TRVDocParameters.UnitsPerInch ... 281
Examples ... 282
Loading UTF-8 files ... 282
TRichView Edit ... 282
Properties ... 287
TRichView Edit.AcceptDragDropFormats, AcceptPasteFormats ... 287
TRichView Edit.ActualCurTextStyleNo ... 288
TRichView Edit.CurItemNo ... 288
TRichView Edit.CurItemStyle ... 289
TRichView Edit.CurParaStyleNo ... 289
TRichView Edit.CurTextStyleNo ... 290
TRichView Edit.CustomCaretInterval ... 290
TRichView Edit.DefaultPictureVAlign ... 291
TRichView Edit.EditorOptions ... 291
TRichView Edit.ForceFieldHighlight ... 293
TRichView Edit.LiveSpellingMode ... 293
TRichView Edit.Modified ... 293
TRichView Edit.OffsetInCurItem ... 294
TRichView Edit.ReadOnly ... 294
TRichView Edit.SmartPopupProperties ... 295
TRichView Edit.SmartPopupVisible ... 295
TRichView Edit.TopLevelEditor ... 295
TRichView Edit.UndoLimit ... 295
Methods ... 296
TRichView Edit.AdjustControlPlacement ... 296
TRichView Edit.AdjustControlPlacement2 ... 296
TRichView Edit.ApplyListStyle ... 297
TRichView Edit.ApplyParaStyle ... 297
TRichView Edit.ApplyParaStyleConversion ... 298
TRichView Edit.ApplyParaStyleTemplate ... 298
TRichView Edit.ApplyStyleConversion ... 299
TRichView Edit.ApplyStyleTemplate ... 299
TRichView Edit.ApplyTextStyle ... 300
TRichView Edit.ApplyTextStyleTemplate ... 301
TRichView Edit.BeginItemModify ... 301
TRichView Edit.BeginUndoCustomGroup ... 302
TRichView Edit.BeginUndoGroup ... 302
TRichView Edit.BeginUpdate ... 302
TRichView Edit.CanChange ... 302
TRichView Edit.CanPaste ... 303

TRichView Edit.CanPasteRTF ... 303
TRichView Edit.CanPasteRVF ... 303
TRichView Edit.Change ... 303
TRichView Edit.ChangeListLevels ... 304
TRichView Edit.ChangeStyleTemplates ... 304
TRichView Edit.Clear ... 304
TRichView Edit.ClearTextFlow ... 305
TRichView Edit.ClearUndo ... 305
TRichView Edit.ConvertToHotPicture ... 305
TRichView Edit.ConvertToPicture ... 305
TRichView Edit.Create ... 306
TRichView Edit.CutDef ... 306
TRichView Edit.DeleteSelection ... 306
TRichView Edit.Destroy ... 306
TRichView Edit.EndItemModify ... 307
TRichView Edit.EndUpdate ... 307
TRichView Edit.GetCheckpointAtCaret ... 307
TRichView Edit.GetCurrentBreakInfo ... 308
TRichView Edit.GetCurrentBulletInfo ... 308
TRichView Edit.GetCurrentCheckpoint ... 309
TRichView Edit.GetCurrentControlInfo ... 309
TRichView Edit.GetCurrentHotspotInfo ... 310
TRichView Edit.GetCurrentItem ... 311
TRichView Edit.GetCurrentItemEx ... 311
TRichView Edit.GetCurrentItemExtraIntProperty -Ex ... 312
TRichView Edit.GetCurrentItemExtraStrProperty -Ex ... 313
TRichView Edit.GetCurrentItemText -A -W ... 313
TRichView Edit.GetCurrentItemVAlign ... 314
TRichView Edit.GetCurrentLineCol ... 314
TRichView Edit.GetCurrentMisspelling ... 315
TRichView Edit.GetCurrentPictureInfo ... 315
TRichView Edit.GetCurrentTag ... 316
TRichView Edit.GetCurrentTextInfo ... 316
TRichView Edit.InsertBreak ... 317
TRichView Edit.InsertBullet ... 317
TRichView Edit.InsertCheckpoint ... 318
TRichView Edit.InsertControl ... 318
TRichView Edit.InsertHotPicture ... 319
TRichView Edit.InsertHotspot ... 319
TRichView Edit.InsertItem ... 320
TRichView Edit.InsertOEMTextFromFile ... 321
TRichView Edit.InsertPageBreak ... 321
TRichView Edit.InsertPicture ... 322
TRichView Edit.InsertRTFFromFileEd ... 323
TRichView Edit.InsertRTFFromStreamEd ... 323
TRichView Edit.InsertRVFFromFileEd ... 324
TRichView Edit.InsertRVFFromStreamEd ... 325
TRichView Edit.InsertStringTag -A -W ... 326
TRichView Edit.InsertTab ... 326
TRichView Edit.InsertText -A -W ... 327
TRichView Edit.InsertTextFromFile -W ... 328
TRichView Edit.Paste ... 328
TRichView Edit.PasteBitmap ... 329
TRichView Edit.PasteGraphicFiles ... 329
TRichView Edit.PasteMetafile ... 330
TRichView Edit.PasteRTF ... 330
TRichView Edit.PasteRVF ... 331
TRichView Edit.PasteText -A -W ... 331

TRichView Edit.PasteURL ... 332
TRichView Edit.Redo ... 332
TRichView Edit.RedoAction ... 333
TRichView Edit.RedoName ... 333
TRichView Edit.RemoveCheckpointAtCaret ... 333
TRichView Edit.RemoveCheckpointEd ... 333
TRichView Edit.RemoveCurrentCheckpoint ... 334
TRichView Edit.RemoveCurrentPageBreak ... 334
TRichView Edit.RemoveLists ... 334
TRichView Edit.ResizeControl ... 335
TRichView Edit.ResizeCurrentControl ... 335
TRichView Edit.SearchText -W ... 336
TRichView Edit.SelectCurrentLine ... 337
TRichView Edit.SelectCurrentWord ... 337
TRichView Edit.Set*PropertyEd ... 337
TRichView Edit.SetBackgroundImageEd ... 338
TRichView Edit.SetBreakInfoEd ... 339
TRichView Edit.SetBulletInfoEd ... 339
TRichView Edit.SetCheckpointInfoEd ... 340
TRichView Edit.SetControlInfoEd ... 341
TRichView Edit.SetCurrentBreakInfo ... 342
TRichView Edit.SetCurrentBulletInfo ... 342
TRichView Edit.SetCurrentCheckpointInfo ... 343
TRichView Edit.SetCurrentControlInfo ... 344
TRichView Edit.SetCurrentHotspotInfo ... 345
TRichView Edit.SetCurrentItemExtraIntProperty -Ex ... 345
TRichView Edit.SetCurrentItemExtraStrProperty -Ex ... 346
TRichView Edit.SetCurrentItemText -A -W ... 347
TRichView Edit.SetCurrentItemVAlign ... 348
TRichView Edit.SetCurrentPictureInfo ... 348
TRichView Edit.SetCurrentTag ... 349
TRichView Edit.SetHotspotInfoEd ... 349
TRichView Edit.SetItemExtraIntProperty-Ed -ExEd ... 350
TRichView Edit.SetItemExtraStrProperty-Ed -ExEd ... 351
TRichView Edit.SetItemTagEd ... 352
TRichView Edit.SetItemTextEd -A -W ... 352
TRichView Edit.SetItemVAlignEd ... 353
TRichView Edit.SetPictureInfoEd ... 354
TRichView Edit.SetUndoGroupMode ... 355
TRichView Edit.Undo ... 355
TRichView Edit.UndoAction ... 355
TRichView Edit.UndoName ... 355
Events ... 356
TRichView Edit.OnCaretGetOut ... 356
TRichView Edit.OnCaretMove ... 356
TRichView Edit.OnChange ... 357
TRichView Edit.OnChanging ... 357
TRichView Edit.OnCheckStickingItems ... 357
TRichView Edit.OnCurParaStyleChanged ... 358
TRichView Edit.OnCurTextStyleChanged ... 358
TRichView Edit.OnDraw CustomCaret ... 358
TRichView Edit.OnDropFiles ... 359
TRichView Edit.OnItemResize ... 359
TRichView Edit.OnItemTextEdit ... 360
TRichView Edit.OnMeasureCustomCaret ... 360
TRichView Edit.OnOleDragEnter ... 361
TRichView Edit.OnOleDragLeave ... 362
TRichView Edit.OnOleDragOver ... 362

TRichView Edit.OnOleDrop ... 363
TRichView Edit.OnParaStyleConversion ... 364
TRichView Edit.OnPaste ... 365
TRichView Edit.OnSmartPopupClick ... 366
TRichView Edit.OnStyleConversion ... 366
Classes of Properties ... 368
TRVSmartPopupProperties ... 368
Properties ... 369
TRVSmartPopupProperties.ButtonType ... 369
TRVSmartPopupProperties.Color ... 369
TRVSmartPopupProperties.Hint ... 370
TRVSmartPopupProperties.HoverColor ... 370
TRVSmartPopupProperties.HoverLineColor ... 370
TRVSmartPopupProperties.ImageIndex ... 370
TRVSmartPopupProperties.ImageList ... 370
TRVSmartPopupProperties.LineColor ... 370
TRVSmartPopupProperties.Menu ... 371
TRVSmartPopupProperties.Position ... 371
TRVSmartPopupProperties.ShortCut ... 371
Methods ... 371
TRVSmartPopupProperties.SetButtonState ... 371
Examples ... 371
Moving caret to the beginning of paragraph ... 371
OnDropFile Example ... 372
TDBRichView ... 373
Properties ... 374
TDBRichView .AutoDisplay ... 374
TDBRichView .DataField ... 375
TDBRichView .DataSource ... 375
TDBRichView .Field ... 375
Methods ... 375
TDBRichView .Create ... 375
TDBRichView .Destroy ... 376
TDBRichView .LoadField ... 376
Events ... 376
TDBRichView .OnLoadCustomFormat ... 376
TDBRichView .OnLoadDocument ... 376
TDBRichView .OnNew Document ... 377
TDBRichView Edit ... 377
Properties ... 378
TDBRichView Edit.AutoDeleteUnusedStyles ... 378
TDBRichView Edit.AutoDisplay ... 379
TDBRichView Edit.DataField ... 379
TDBRichView Edit.DataSource ... 379
TDBRichView Edit.Field ... 379
TDBRichView Edit.FieldFormat ... 380
TDBRichView Edit.IgnoreEscape ... 380
TDBRichView Edit.ReadOnly ... 380
Methods ... 381
TDBRichView Edit.Change ... 381
TDBRichView Edit.Create ... 381
TDBRichView Edit.Destroy ... 381
TDBRichView Edit.LoadField ... 381
Events ... 381
TDBRichView Edit.OnLoadCustomFormat ... 381
TDBRichView Edit.OnLoadDocument ... 382
TDBRichView Edit.OnNew Document ... 382
TDBRichView Edit.OnSaveCustomFormat ... 382

Examples ... 383
Using viewer-style methods in DBRichViewEdit ... 383
Inserting controls what can modify themselves in DBRichViewEdit ... 383
TRVPrintPreview ... 384
Properties ... 384
TRVPrintPreview.RVPrint ... 384
Additional Components ... 385
TRVStyle ... 385
Properties ... 387
TRVStyle.CheckpointColor, CheckpointColor ... 387
TRVStyle.Color ... 387
TRVStyle.CurrentItemColor ... 388
TRVStyle.DefCodePage ... 388
TRVStyle.DefTabWidth ... 388
TRVStyle.DefUnicodeStyle ... 389
TRVStyle.DisabledFontColor ... 389
TRVStyle.EndnoteNumbering ... 389
TRVStyle.FieldHighlightColor ... 390
TRVStyle.FieldHighlightType ... 390
TRVStyle.FloatingLineColor ... 390
TRVStyle.FontQuality ... 391
TRVStyle.FootnotePageReset ... 392
TRVStyle.FullRedraw ... 392
TRVStyle.GridColor, GridReadOnlyColor, GridStyle, GridReadOnlyStyle ... 392
TRVStyle.HoverColor ... 393
TRVStyle.InactiveSelColor, InactiveSelTextColor, SelColor, SelTextColor ... 393
TRVStyle.InvalidPicture ... 393
TRVStyle.JumpCursor ... 394
TRVStyle.LineSelectCursor ... 394
TRVStyle.LineWrapMode ... 394
TRVStyle.ListStyles ... 396
TRVStyle.LiveSpellingColor ... 397
TRVStyle.MainRVStyle ... 397
TRVStyle.PageBreakColor ... 398
TRVStyle.ParaStyles ... 398
TRVStyle.SelectionHandleKind ... 398
TRVStyle.SelectionMode ... 399
TRVStyle.SelectionStyle ... 399
TRVStyle.SoftPageBreakColor ... 401
TRVStyle.SpacesInTab ... 402
TRVStyle.SpecialCharactersColor ... 402
TRVStyle.StyleTemplates ... 402
TRVStyle.TextBackgroundKind ... 402
TRVStyle.TextEngine ... 403
TRVStyle.TextStyles ... 404
TRVStyle.Units ... 404
TRVStyle.UseSound ... 404
Methods ... 405
TRVStyle.AddTextStyle ... 405
TRVStyle.Create ... 405
TRVStyle.ConvertTo* ... 405
TRVStyle.DeleteTextStyle ... 405
TRVStyle.Destroy ... 405
TRVStyle.GetAs* ... 406
TRVStyle.*ToUnits ... 406
TRVStyle.GetListStyleClass ... 406
TRVStyle.GetParaStyleClass ... 407
TRVStyle.GetTextStyleClass ... 407

TRVStyle.LoadINI ... 407
TRVStyle.FindTextStyle, FindParaStyle ... 407
TRVStyle.LoadReg ... 408
TRVStyle.SaveCSS ... 408
TRVStyle.SaveCSSToStream ... 409
TRVStyle.SaveINI ... 409
TRVStyle.SaveReg ... 409
Events ... 410
TRVStyle.OnAfterApplyStyle ... 410
TRVStyle.OnApplyStyle ... 410
TRVStyle.OnApplyStyleColor ... 411
TRVStyle.OnDrawCheckpoint ... 411
TRVStyle.OnDrawPageBreak ... 413
TRVStyle.OnDrawParaBack ... 414
TRVStyle.OnDrawStyleText ... 415
TRVStyle.OnDrawTextBack ... 416
TRVStyle.OnStyleHoverSensitive ... 417
Classes of Properties - Style Collections ... 418
TFontInfos ... 418
Properties ... 418
TFontInfos.InvalidItem ... 418
TFontInfos.Items ... 418
TFontInfos.PixelsPerInch ... 419
Methods ... 419
TFontInfos.Add ... 419
TFontInfos.AddFont ... 419
TFontInfos.AddFontEx ... 419
TFontInfos.AssignTo ... 419
TFontInfos.Create ... 420
TFontInfos.FindStyleWithCharset ... 420
TFontInfos.FindStyleWithColor ... 420
TFontInfos.FindStyleWithFont ... 420
TFontInfos.FindStyleWithFontName ... 421
TFontInfos.FindStyleWithFontSize ... 421
TFontInfos.FindStyleWithFontStyle ... 421
TFontInfos.FindSuchStyle, FindSuchStyleEx ... 422
TParaInfos ... 423
Properties ... 423
TParaInfos.InvalidItem ... 423
TParaInfos.Items ... 423
Methods ... 424
TParaInfos.Add ... 424
TParaInfos.AssignTo ... 424
TParaInfos.Create ... 424
TParaInfos.FindStyleWithAlignment ... 424
TParaInfos.FindSuchStyle, FindSuchStyleEx ... 425
TRVListInfos ... 425
Properties ... 426
TRVListInfos.Items ... 426
Methods ... 426
TRVListInfos.Add ... 426
TRVListInfos.AssignTo ... 426
TRVListInfos.FindStyleWithLevels ... 426
TRVListInfos.FindSuchStyle ... 426
TRVStyleTemplateCollection ... 427
Properties ... 427
TRVStyleTemplateCollection.DefStyleName ... 427
TRVStyleTemplateCollection.ForbiddenIds ... 428

TRVStyleTemplateCollection.Items ... 428
TRVStyleTemplateCollection.NormalStyleTemplate ... 428
Methods ... 428
TRVStyleTemplateCollection.AssignStyleTemplates ... 428
TRVStyleTemplateCollection.AssignToStrings ... 428
TRVStyleTemplateCollection.ClearTextFormat, ClearParaFormat ... 429
TRVStyleTemplateCollection.FindById, FindItemById ... 429
TRVStyleTemplateCollection.FindByName, FindItemByName ... 429
TRVStyleTemplateCollection.LoadFromRVST, SaveToRVST ... 429
TRVStyleTemplateCollection.ResetNameCounter ... 430
Classes of Properties - Styles ... 430
TCustomRVInfo, TCustomRVTextOrParaInfo ... 430
Properties ... 431
TCustomRVInfo.Standard ... 431
TCustomRVInfo.StyleName ... 432
TCustomRVFontOrParaInfo.StyleTemplateId ... 432
TCustomRVFontInfo ... 432
Properties ... 433
TCustomRVFontInfo.BackColor ... 433
TCustomRVFontInfo.BiDiMode ... 433
TCustomRVFontInfo.CharScale ... 434
TCustomRVFontInfo.Charset ... 434
TCustomRVFontInfo.CharSpacing ... 435
TCustomRVFontInfo.Color ... 435
TCustomRVFontInfo.FontName ... 436
TCustomRVFontInfo.HoverBackColor ... 436
TCustomRVFontInfo.HoverColor ... 436
TCustomRVFontInfo.HoverEffects ... 437
TCustomRVFontInfo.HoverUnderlineColor ... 437
TCustomRVFontInfo.JumpCursor ... 438
TCustomRVFontInfo.Options ... 438
TCustomRVFontInfo.ParaStyleTemplateId ... 439
TCustomRVFontInfo.Protection ... 439
TCustomRVFontInfo.Size ... 441
TCustomRVFontInfo.Style ... 441
TCustomRVFontInfo.StyleEx ... 441
TCustomRVFontInfo.SubSuperScriptType ... 441
TCustomRVFontInfo.UnderlineColor ... 442
TCustomRVFontInfo.UnderlineType ... 442
TCustomRVFontInfo.VShift ... 443
Methods ... 443
TCustomRVFontInfo.Assign ... 443
TCustomRVFontInfo.AssignTo ... 443
TCustomRVFontInfo.Create ... 444
TCustomRVFontInfo.IsEqual ... 444
TFontInfo ... 444
Properties ... 445
TFontInfo.Jump ... 445
TFontInfo.ModifiedProperties ... 445
TFontInfo.NextStyleNo ... 446
TFontInfo.Unicode ... 446
Methods ... 446
TFontInfo.Assign ... 446
TFontInfo.Create ... 447
TFontInfo.Draw ... 447
TFontInfo.IsEqual ... 448
TCustomRVParaInfo ... 448
Properties ... 449

TCustomRVParaInfo.Alignment, LastLineAlignment ... 449
TCustomRVParaInfo.Background ... 450
TCustomRVParaInfo.BiDiMode ... 450
TCustomRVParaInfo.Border ... 451
TCustomRVParaInfo.FirstIndent ... 451
TCustomRVParaInfo.LeftIndent ... 451
TCustomRVParaInfo.LineSpacing ... 452
TCustomRVParaInfo.LineSpacingType ... 452
TCustomRVParaInfo.Options ... 453
TCustomRVParaInfo.OutlineLevel ... 454
TCustomRVParaInfo.RightIndent ... 454
TCustomRVParaInfo.SpaceAfter ... 454
TCustomRVParaInfo.SpaceBefore ... 455
TCustomRVParaInfo.Tabs ... 455
Methods ... 455
TCustomRVParaInfo.Assign ... 455
TCustomRVParaInfo.Create ... 455
TCustomRVParaInfo.IsEqual ... 456
TParaInfo ... 456
Properties ... 457
TParaInfo.DefStyleNo ... 457
TParaInfo.ModifiedProperties ... 457
TParaInfo.NextParaNo ... 457
Methods ... 458
TParaInfo.Assign ... 458
TParaInfo.Create ... 458
TParaInfo.IsEqual ... 458
TRVListInfo ... 459
Properties ... 459
TRVListInfo.Levels ... 459
TRVListInfo.OneLevelPreview ... 459
Methods ... 459
TRVListInfo.AllNumbered ... 459
TRVListInfo.HasNumbering ... 460
TRVStyleTemplate ... 460
Properties ... 461
TRVStyleTemplate.Id ... 461
TRVStyleTemplate.Kind ... 461
TRVStyleTemplate.Name ... 461
TRVStyleTemplate.NextId, Next ... 462
TRVStyleTemplate.ParaStyle ... 463
TRVStyleTemplate.ParentId, Parent ... 463
TRVStyleTemplate.QuickAccess ... 463
TRVStyleTemplate.TextStyle ... 463
TRVStyleTemplate.ValidParaProperties ... 463
TRVStyleTemplate.ValidTextProperties ... 464
Methods ... 464
TRVStyleTemplate.ApplyToParaStyle ... 464
TRVStyleTemplate.ApplyToTextStyle ... 465
TRVStyleTemplate.UpdateModifiedTextStyleProperties, UpdateModifiedParaStyleProperties ... 466
Classes of Properties of Paragraph Style ... 466
TRVBackgroundRect ... 466
Properties ... 466
TRVBackgroundRect.BorderOffsets ... 466
TRVBackgroundRect.Color ... 467
TRVBackgroundRect.Opacity ... 467
TRVBorder ... 467
Properties ... 467

TRVBorder.BorderOffsets ... 467
TRVBorder.Color ... 468
TRVBorder.InternalWidth ... 468
TRVBorder.Style ... 468
TRVBorder.VisibleBorders ... 468
TRVBorder.Width ... 468
TRVTabInfos ... 469
Properties ... 469
TRVTabInfos.Items ... 469
Methods ... 469
TRVTabInfos.AddFrom ... 469
TRVTabInfos.DeleteList ... 469
TRVTabInfos.Find ... 469
TRVTabInfos.Intersect ... 470
TRVTabInfo ... 470
Properties ... 470
TRVTabInfo.Align ... 470
TRVTabInfo.Leader ... 470
TRVTabInfo.Position ... 471
Classes of Properties of List Style ... 471
TRVListLevelCollection ... 471
Properties ... 472
TRVListLevelCollection.Items ... 472
Methods ... 472
TRVListLevelCollection.Add ... 472
TRVListLevel ... 472
Properties ... 473
TRVListLevel.FirstIndent ... 473
TRVListLevel.Font ... 473
TRVListLevel.FormatString ... 474
TRVListLevel.FormatStringW ... 475
TRVListLevel.ImageIndex ... 475
TRVListLevel.ImageList ... 475
TRVListLevel.LeftIndent ... 476
TRVListLevel.ListType ... 476
TRVListLevel.MarkerAlignment ... 477
TRVListLevel.MarkerIndent ... 478
TRVListLevel.Options ... 478
TRVListLevel.Picture ... 479
TRVListLevel.StartFrom ... 479
Methods ... 479
TRVListLevel.HasNumbering ... 479
TRVPrint ... 479
Properties ... 484
TRVPrint.ClipMargins ... 484
TRVPrint.FacingPages ... 484
TRVPrint.FixMarginsMode ... 484
TRVPrint.FooterY ... 485
TRVPrint.HeaderY ... 485
TRVPrint.Margins ... 486
TRVPrint.MirrorMargins ... 487
TRVPrint.Preview100PercentHeight ... 487
TRVPrint.Preview100PercentWidth ... 487
TRVPrint.TitlePage ... 487
TRVPrint.Units ... 487
Methods ... 488
TRVPrint.AssignDocParameters ... 488
TRVPrint.AssignSource ... 488

TRVPrint.ContinuousPrint ... 488
TRVPrint.ConvertToUnits ... 489
TRVPrint.Create ... 489
TRVPrint.Draw Preview ... 489
TRVPrint.FormatPages ... 490
TRVPrint.GetFooterRect ... 490
TRVPrint.GetHeaderRect ... 490
TRVPrint.MakePreview ... 491
TRVPrint.MakeScaledPreview ... 491
TRVPrint.Print ... 492
TRVPrint.PrintPages ... 492
TRVPrint.SetFooter ... 493
TRVPrint.SetHeader ... 493
Events ... 494
TRVPrint.OnFormatting ... 494
TRVPrint.OnPagePostpaint ... 495
TRVPrint.OnPagePrepaint ... 495
TRVPrint.OnSendingToPrinter ... 496
TRVOfficeConverter ... 496
Properties ... 497
TRVOfficeConverter.ErrorCode ... 497
TRVOfficeConverter.ExcludeDocX-Converter ... 498
TRVOfficeConverter.ExcludeHTML-Converter ... 498
TRVOfficeConverter.ExportConverters ... 499
TRVOfficeConverter.ExtensionsInFilter ... 499
TRVOfficeConverter.ImportConverters ... 499
TRVOfficeConverter.Preview Mode ... 499
TRVOfficeConverter.Stream ... 500
Methods ... 500
TRVOfficeConverter.Create ... 500
TRVOfficeConverter.Destroy ... 500
TRVOfficeConverter.ExportRTF ... 500
TRVOfficeConverter.ExportRV ... 500
TRVOfficeConverter.GetExportFilter ... 501
TRVOfficeConverter.GetImportFilter ... 501
TRVOfficeConverter.ImportRTF ... 501
TRVOfficeConverter.ImportRV ... 502
TRVOfficeConverter.IsValidImporter ... 502
Events ... 503
TRVOfficeConverter.OnConverting ... 503
Classes of Properties ... 503
TRVOfficeCnvList ... 503
TRVOfficeConverterInfo ... 503
TRVReportHelper ... 503
Properties ... 504
TRVReportHelper.Allow TransparentDraw ing ... 504
TRVReportHelper.NoMetafiles ... 505
TRVReportHelper.RichView ... 505
Methods ... 505
TRVReportHelper.Draw Page ... 505
TRVReportHelper.Draw PageAt ... 506
TRVReportHelper.Finished ... 506
TRVReportHelper.FormatNextPage ... 506
TRVReportHelper.GetLastPageHeight ... 506
TRVReportHelper.Init ... 507
TRVReportHelper.UpdatePageNumbers ... 507
Events ... 507
TRVReportHelper.OnDraw Checkpoint ... 507

TRVReportHelper.OnDrawHyperlink ... 508
TRVReportHelper.OnGetPageNumber ... 508
Components Ancestor Classes ... 508
TRVScroller ... 508
Properties ... 509
TRVScroller.BorderStyle ... 509
TRVScroller.HScrollMax ... 509
TRVScroller.HScrollPos ... 510
TRVScroller.HScrollVisible ... 510
TRVScroller.InplaceEditor ... 510
TRVScroller.NoVScroll ... 511
TRVScroller.Tracking ... 511
TRVScroller.UseXPThemes ... 511
TRVScroller.VScrollMax ... 512
TRVScroller.VScrollPos ... 512
TRVScroller.VScrollVisible ... 512
TRVScroller.WheelStep ... 512
Methods ... 513
TRVScroller.Create ... 513
TRVScroller.ScrollTo ... 513
Events ... 513
TRVScroller.OnHScrolled ... 513
TRVScroller.OnVScrolled ... 513
TCustomRVPrint ... 514
Properties ... 514
TCustomRVPrint.ColorMode ... 514
TCustomRVPrint.EndAt ... 514
TCustomRVPrint.FirstPageNo ... 514
TCustomRVPrint.IgnorePageBreaks ... 515
TCustomRVPrint.LastPageNo ... 515
TCustomRVPrint.MaxPrintedItemNo ... 515
TCustomRVPrint.MetafileCompatibility ... 515
TCustomRVPrint.MinPrintedItemNo ... 516
TCustomRVPrint.PageNoFromNumber ... 516
TCustomRVPrint.PagesCount ... 516
TCustomRVPrint.PreviewCorrection ... 516
TCustomRVPrint.StartAt ... 517
TCustomRVPrint.TransparentBackground ... 517
Methods ... 517
TCustomRVPrint.Clear ... 517
TCustomRVPrint.GetFirstItemOnPage ... 517
TCustomRVPrint.GetPageNo ... 518
TCustomRVPrint.UpdatePaletteInfo ... 518
Events ... 519
TCustomRVPrint.OnPrintComponent ... 519
TCustomRVPrintPreview ... 520
Properties ... 521
TCustomRVPrintPreview.ClickMode ... 521
TCustomRVPrintPreview.Color ... 521
TCustomRVPrintPreview.MarginsPen ... 522
TCustomRVPrintPreview.PageBorderColor ... 522
TCustomRVPrintPreview.PageBorderWidth ... 522
TCustomRVPrintPreview.PageNo ... 522
TCustomRVPrintPreview.PrintableAreaPen ... 523
TCustomRVPrintPreview.ShadowColor ... 523
TCustomRVPrintPreview.ShadowWidth ... 523
TCustomRVPrintPreview.ZoomInCursor ... 523
TCustomRVPrintPreview.ZoomMode ... 524

TCustomRVPrintPreview.ZoomOutCursor ... 524
TCustomRVPrintPreview.ZoomPercent ... 525
Methods ... 525
TCustomRVPrintPreview.Create ... 525
TCustomRVPrintPreview.First ... 525
TCustomRVPrintPreview.Last ... 525
TCustomRVPrintPreview.Next ... 526
TCustomRVPrintPreview.Prev ... 526
TCustomRVPrintPreview.SetZoom ... 526
Events ... 526
TCustomRVPrintPreview.OnZoomChanged ... 526
.5..Item Types ... 526
Table - TRVTableItemInfo ... 528
Properties ... 530
TRVTableItemInfo.BackgroundImage ... 530
TRVTableItemInfo.BackgroundImageFileName ... 531
TRVTableItemInfo.BackgroundStyle ... 531
TRVTableItemInfo.BestWidth ... 531
TRVTableItemInfo.BorderColor ... 532
TRVTableItemInfo.BorderHSpacing ... 532
TRVTableItemInfo.BorderLightColor ... 532
TRVTableItemInfo.BorderStyle ... 533
TRVTableItemInfo.BorderVSpacing ... 533
TRVTableItemInfo.BorderWidth ... 534
TRVTableItemInfo.CellBorderColor ... 534
TRVTableItemInfo.CellBorderLightColor ... 534
TRVTableItemInfo.CellBorderStyle ... 535
TRVTableItemInfo.CellBorderWidth ... 535
TRVTableItemInfo.CellHPadding ... 536
TRVTableItemInfo.CellHSpacing ... 536
TRVTableItemInfo.CellPadding ... 536
TRVTableItemInfo.Cells ... 537
TRVTableItemInfo.CellVPadding ... 537
TRVTableItemInfo.CellVSpacing ... 537
TRVTableItemInfo.ColBandSize, RowBandSize ... 538
TRVTableItemInfo.ColCount ... 538
TRVTableItemInfo.Color ... 538
TRVTableItemInfo.HeadingRowCount ... 539
TRVTableItemInfo.HOutermostRule ... 539
TRVTableItemInfo.HRuleColor ... 539
TRVTableItemInfo.HRuleWidth ... 540
TRVTableItemInfo.Opacity ... 540
TRVTableItemInfo.Options ... 540
TRVTableItemInfo.PrintOptions ... 542
TRVTableItemInfo.RowCount ... 543
TRVTableItemInfo.Rows ... 543
TRVTableItemInfo.TextColSeparator ... 543
TRVTableItemInfo.TextRowSeparator ... 544
TRVTableItemInfo.VisibleBorders ... 544
TRVTableItemInfo.VOutermostRule ... 545
TRVTableItemInfo.VRuleColor ... 545
TRVTableItemInfo.VRuleWidth ... 545
TRVTableItemInfo's row and column colors ... 546
Methods ... 547
TRVTableItemInfo.AssignProperties ... 547
TRVTableItemInfo.CanMergeCells ... 547
TRVTableItemInfo.CanMergeSelectedCells ... 548

TRVTableItemInfo.Changed ... 548
TRVTableItemInfo.Create ... 548
TRVTableItemInfo.CreateEx ... 549
TRVTableItemInfo.DeleteCols ... 549
TRVTableItemInfo.DeleteEmptyCols ... 550
TRVTableItemInfo.DeleteEmptyRows ... 550
TRVTableItemInfo.DeleteRows ... 550
TRVTableItemInfo.DeleteSelectedCols ... 551
TRVTableItemInfo.DeleteSelectedRows ... 551
TRVTableItemInfo.Deselect ... 551
TRVTableItemInfo.Destroy ... 551
TRVTableItemInfo.EditCell ... 552
TRVTableItemInfo.GetCellAt ... 552
TRVTableItemInfo.GetEditedCell ... 552
TRVTableItemInfo.GetNormalizedSelectionBounds ... 553
TRVTableItemInfo.GetSelectionBounds ... 553
TRVTableItemInfo.InsertCols ... 554
TRVTableItemInfo.InsertColsLeft ... 554
TRVTableItemInfo.InsertColsRight ... 554
TRVTableItemInfo.InsertRows ... 555
TRVTableItemInfo.InsertRowsAbove ... 555
TRVTableItemInfo.InsertRowsBelow ... 555
TRVTableItemInfo.IsCellSelected ... 556
TRVTableItemInfo.LoadFromStream ... 556
TRVTableItemInfo.MergeCells ... 556
TRVTableItemInfo.MergeSelectedCells ... 557
TRVTableItemInfo.SaveRowsToStream ... 557
TRVTableItemInfo.SaveToStream ... 557
TRVTableItemInfo.Select ... 558
TRVTableItemInfo.SelectCols ... 558
TRVTableItemInfo.SelectRows ... 558
TRVTableItemInfo.SetCellBackgroundImage ... 559
TRVTableItemInfo.SetCellBackgroundImageFileName ... 559
TRVTableItemInfo.SetCellBackgroundStyle ... 559
TRVTableItemInfo.SetCellBestHeight ... 559
TRVTableItemInfo.SetCellBestWidth ... 560
TRVTableItemInfo.SetCellBorderColor ... 560
TRVTableItemInfo.SetCellBorderLightColor ... 560
TRVTableItemInfo.SetCellColor ... 560
TRVTableItemInfo.SetCellHint ... 561
TRVTableItemInfo.SetCellOpacity ... 561
TRVTableItemInfo.SetCellOptions ... 561
TRVTableItemInfo.SetCellRotation ... 561
TRVTableItemInfo.SetCellTag ... 562
TRVTableItemInfo.SetCellVAlign ... 562
TRVTableItemInfo.SetCellVisibleBorders ... 562
TRVTableItemInfo.SetRow* ... 562
TRVTableItemInfo.SetTableVisibleBorders ... 563
TRVTableItemInfo.SplitSelectedCellsHorizontally ... 563
TRVTableItemInfo.SplitSelectedCellsVertically ... 563
TRVTableItemInfo.UnmergeCells ... 564
TRVTableItemInfo.UnmergeSelectedCells ... 564
Events ... 564
TRVTableItemInfo.OnCellEditing ... 564
TRVTableItemInfo.OnDrawBorder ... 565
Classes of Properties ... 567
TRVTableCellData ... 567
Properties ... 567

TRVTableCellData.BackgroundImage ... 567
TRVTableCellData.BackgroundImageFileName ... 568
TRVTableCellData.BackgroundStyle ... 568
TRVTableCellData.BestHeight ... 568
TRVTableCellData.BestWidth ... 569
TRVTableCellData.BorderColor ... 569
TRVTableCellData.BorderLightColor ... 570
TRVTableCellData.Color ... 570
TRVTableCellData.ColSpan ... 570
TRVTableCellData.Hint ... 571
TRVTableCellData.Opacity ... 571
TRVTableCellData.Options ... 571
TRVTableCellData.Rotation ... 573
TRVTableCellData.Row Span ... 574
TRVTableCellData.Tag ... 574
TRVTableCellData.VAlign ... 575
TRVTableCellData.VisibleBorders ... 575
Methods ... 575
TRVTableCellData.GetRVData ... 575
Additional properties of TRVTableCellData ... 576
TRVTableRow ... 577
Properties ... 577
TRVTableRow .Items ... 577
TRVTableRow .KeepTogether ... 577
TRVTableRow .PageBreakBefore ... 578
TRVTableRow .VAlign ... 578
TRVTableRow s ... 578
Properties ... 579
TRVTableRow s.Items ... 579
Methods ... 579
TRVTableRow s.GetMainCell ... 579
Label - TRVLabelItemInfo ... 579
Properties ... 580
TRVLabelItemInfo.Alignment ... 580
TRVLabelItemInfo.Cursor ... 580
TRVLabelItemInfo.MinWidth ... 580
TRVLabelItemInfo.ProtectTextStyleNo ... 581
TRVLabelItemInfo.RemoveInternalLeading ... 581
TRVLabelItemInfo.Text ... 581
TRVLabelItemInfo.TextStyleNo ... 581
Methods ... 582
TRVLabelItemInfo.Create ... 582
TRVLabelItemInfo.CreateEx ... 582
Numbered Sequence - TRVSeqItemInfo ... 582
Properties ... 584
TRVSeqItemInfo.FormatString ... 584
TRVSeqItemInfo.NumberType ... 584
TRVSeqItemInfo.Reset ... 584
TRVSeqItemInfo.SeqName ... 585
TRVSeqItemInfo.StartFrom ... 585
Methods ... 585
TRVSeqItemInfo.Create ... 585
TRVSeqItemInfo.CreateEx ... 585
Ancestor for Notes - TCustomRVNoteItemInfo ... 586
Properties ... 586
TCustomRVNoteItemInfo.Document ... 586
TCustomRVNoteItemInfo.NoteText ... 587
Methods ... 587

TCustomRVNoteItemInfo.CreateEx ... 587
TCustomRVNoteItemInfo.ReplaceDocumentEd ... 587
Endnote - TRVEndnoteItemInfo ... 587
Methods ... 588
TRVEndnoteItemInfo.Create ... 588
TRVEndnoteItemInfo.CreateEx ... 589
Footnote - TRVFootnoteItemInfo ... 589
Methods ... 590
TRVFootnoteItemInfo.Create ... 590
TRVFootnoteItemInfo.CreateEx ... 591
Sidenote - TRVSidenoteItemInfo ... 591
Properties ... 592
TRVSidenoteItemInfo.BoxProperties ... 592
TRVSidenoteItemInfo.BoxPosition ... 592
Methods ... 592
TRVSidenoteItemInfo.Create ... 592
TRVSidenoteItemInfo.CreateEx ... 593
Classes of Properties ... 593
TRVBoxProperties ... 593
Properties ... 594
TRVBoxProperties.Background ... 594
TRVBoxProperties.Border ... 594
TRVBoxProperties.Height, HeightType ... 594
TRVBoxProperties.VAlign ... 595
TRVBoxProperties.Width, WidthType ... 595
TRVBoxPosition ... 596
Properties ... 596
TRVBoxPosition.HorizontalAnchor ... 596
TRVBoxPosition.HorizontalXXX ... 597
TRVBoxPosition.PositionInText ... 599
TRVBoxPosition.RelativeToCell ... 599
TRVBoxPosition.VerticalAnchor ... 599
TRVBoxPosition.VerticalXXX ... 600
TextBox - TRVTextBoxItemInfo ... 601
Methods ... 602
TRVTextBoxItemInfo.Create ... 602
TRVTextBoxItemInfo.CreateEx ... 602
Reference to the Parent Note - TRVNoteReferenceItemInfo ... 602
Methods ... 604
TRVNoteReferenceItemInfo.Create ... 604
TRVNoteReferenceItemInfo.CreateEx ... 604
Ancestor for Text Fields - TCustomRVFieldItemInfo ... 604
Properties ... 605
TCustomRVPageNumberItemInfo.NumberType ... 605
Page Number - TRVPageNumberItemInfo ... 605
Methods ... 606
TRVPageNumberItemInfo.Create ... 606
TRVPageNumberItemInfo.CreateEx ... 606
Page Count - TRVPageCountItemInfo ... 607
Methods ... 607
TRVPageCountItemInfo.Create ... 607
TRVPageCountItemInfo.CreateEx ... 607
.6 Other Classes ... 608
ERichViewError ... 608
TCustomRVData and Others (TRichView Documents) ... 608
TCustomRVData.Edit ... 610
TCustomRVData.GetRVData ... 611

TCustomRVData.GetSourceRVData ... 611
TCustomRVFormattedData.GetOriginEx ... 611
TPrintableRV ... 611
TRVBooleanRect ... 611
TRVDeleteUnusedStylesData ... 612
TRVGraphicHandler ... 613
TRVIntegerList ... 614
TRVRect ... 615
TRVUnitsRect ... 615
TRVThumbnailMaker ... 615
.7 Procedures and Functions ... 616
DrawControl and Others ... 616
GetRVESearchOptions ... 616
GetRVSearchOptions ... 616
RV_GetPrinterDC ... 617
RVGetFirstEndnote and Others ... 618
RVGetNoteTextStyleNo ... 619
RVIsURL, RVIsEmail ... 619
Unicode Conversion ... 620
Functions from RVGetText Unit ... 620
Functions from RVLinear Unit ... 621
Functions for TRVUnits conversion ... 622
.8 Types ... 622
TCheckpointData ... 622
TRVAnsiString ... 623
TRVBiDiMode ... 623
TRVBorderStyle ... 623
TRVBreakStyle ... 624
TRVCellVAlign ... 625
TRVCodePage ... 625
TRVColorMode ... 625
TRVDisplayOptions ... 626
TRVDocRotation ... 626
TRVESearchOptions ... 626
TRVExtraItemProperty ... 627
TRVExtraItemStrProperty ... 630
TRVFloatPosition ... 632
TRVFloatPositionKind ... 632
TRVFloatProperty ... 632
TRVFloatSize ... 633
TRVFontInfoProperties ... 633
TRVFReaderStyleMode ... 633
TRVHFType ... 634
TRVHTMLLength ... 635
TRVIntProperty ... 635
TRVItemBackgroundStyle ... 636
TRVLength ... 636
TRVOleDropEffects ... 636
TRVOpacity ... 637
TRVPageNumberType ... 637
TRVParaInfoProperties ... 637
TRVPrintingStep ... 638
TRVRawByteString ... 638
TRVReaderStyleMode ... 639
TRVSaveFormat ... 639
TRVSaveOptions ... 639
TRVSearchOptions ... 642

TRVSeqType ... 643
TRVStrProperty ... 643
TRVStyleLength ... 644
TRVStyleTemplateId ... 644
TRVStyleTemplateName ... 644
TRVStyleUnits ... 644
TRVTableBorderStyle ... 645
TRVTag ... 645
TRVTextDrawStates ... 645
TRVUndoType ... 647
TRVUnicodeString ... 647
TRVUnits ... 648
TRVVAlign ... 648
9. Global Constants and Variables ... 650
Clipboard Related Constants and Variables ... 651
Constants Related to HTML and Text ... 652
Constants Related to the Caret ... 652
RichViewAllowCopyTableCells ... 652
RichViewAlternativePicPrint ... 652
RichViewApostropheInWord ... 653
RichViewAutoAssignNormalStyleTemplate ... 653
RichViewCompareStyleNames ... 653
RichViewDoNotCheckRVFStyleRefs ... 653
RichViewDoNotMergeNumbering ... 654
RichViewEditDefaultProportionalResize ... 654
RichViewEditEnterAllowsEmptyMarkeredLines ... 654
RichViewJustifyBeforeLineBreak ... 654
RichViewMaxThumbnailCount ... 655
RichViewPixelsPerInch ... 655
RichViewResetStandardFlag ... 656
RichViewSaveHyperlinksInDocXAsFields ... 656
RichViewShowGhostSpaces ... 656
RichViewStringFormatMethod ... 656
RichViewTableAutoAddRow ... 657
RichViewTableDefaultRTFAutofit ... 657
RichViewUnicodeInput ... 657
rveipcXXX constants ... 658
rvespcXXX constants ... 661
RVGraphicHandler ... 662
RVIsCustomURL ... 662
RVParagraphMarks ... 662
rvsXXX constants ... 663
RVThumbnailMaker ... 664
RVVisibleSpecialCharacters ... 665
10. How To... ... 665
change page size and orientation ... 666
draw page numbers, headers and footers ... 666
implement commands like "make bold", "apply font", etc. ... 667
implement Find and Replace dialogs ... 667
switch insert/overtype mode ... 668
move the caret to the beginning or to the end of document in editor ... 668
make Unicode editor ... 669
make a plain text editor ... 669
implement smart indenting ... 671
combine several TRichView documents in one ... 671
use third-party graphic classes with RichView ... 671
create a chat window ... 673

remove text formatting ... 673
11 Version History ... 675
New after version 16 ... 675
New in version 16 ... 676
New in version 15 ... 678
New in version 14 ... 679
New in version 13 ... 682
New in version 12 ... 684
New in version 11 ... 685
New in version 10 ... 687
New in version 1.9 ... 691
New in version 1.8 ... 693
New in version 1.7 ... 695
New in version 1.6 ... 698
New in version 1.5 ... 700
New in version 1.4 ... 702
New in version 1.3 ... 703
New in version 1.2 ... 704
Compatibility with older versions ... 705
Index 707

1 TRichView Reference

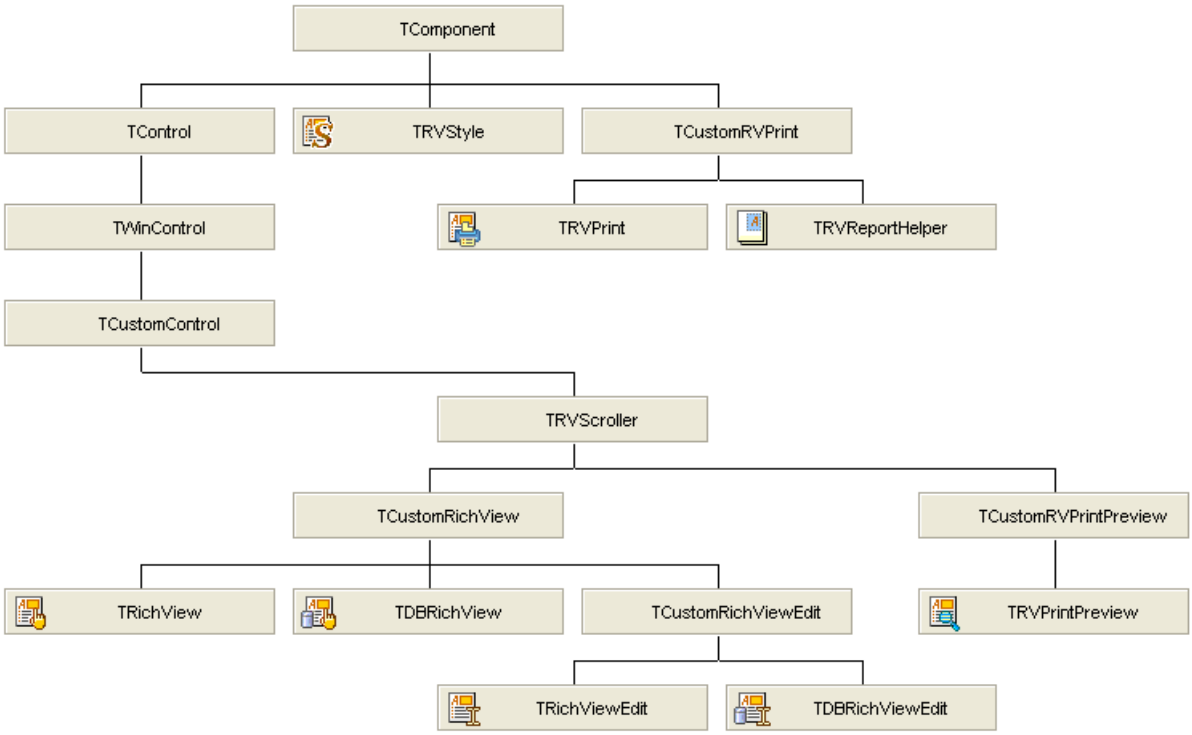
Copyright © Sergey Tkachenko (*www.trichview.com*)
(*svt@trichview.com*)

Version 16 (what's new? 676)

Components

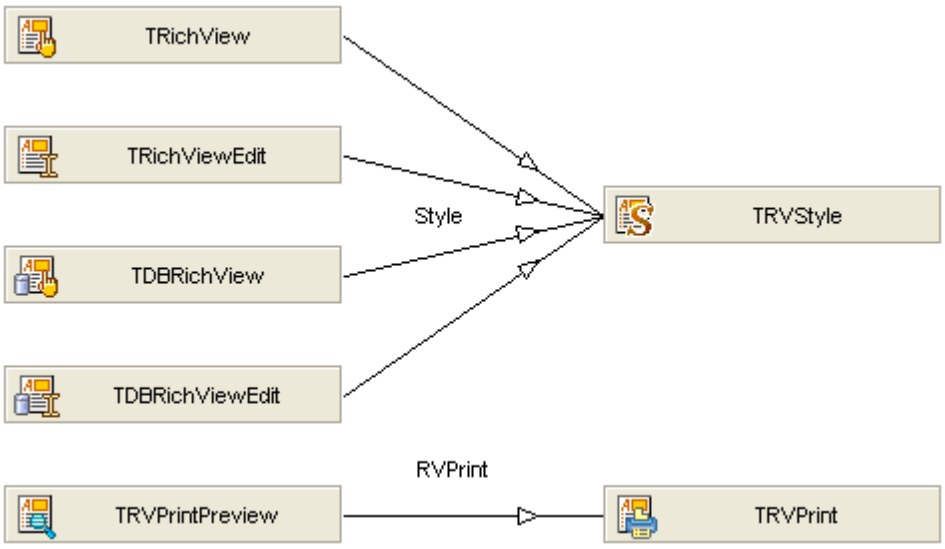
Icon	Class Name	Description
	TRVStyle 385	Component for defining visual appearance of RichView, RichViewEdit, DBRichView, DBRichViewEdit. It contains some properties which can be used by several RichViews.
	TRichView 128	Component for displaying hypertext documents (containing pictures, controls, tables)
	TRichViewEdit 282	Component for editing hypertext documents (containing pictures, controls, tables)
	TDBRichView 373	Data-aware version of RichView
	TDBRichViewEdit 377	Data-aware version of RichViewEdit
	TRVPrint 479	Component for printing documents contained in the components above
	TRVPrintPreview 384	Component allowing to show document as it will be printed by RVPrint
	TRVReportHelper 503	Component for drawing documents on different canvas (screen, printer, image, etc.)
	TRVOfficeConverter 496	Component allowing to use Microsoft Office text converters (import and export in different file formats)

Class Hierarchy



Hierarchy of TRichView classes

Diagram of Using



How TRichView classes use each other

Related topics

- version history and compatibility 675
- overview of features 28
- overview of item types 85 (documents consist of items)
- overview of tables 111

- how to... 665

1.1 Overview

Parts of Documents

- RichView Item Types Overview 29
- RichView Items' "Checkpoints" Overview 30
- RichView Items' "Tags" Overview 34
- RichView Hypertext Overview 35
- RichView Paragraphs 37
- Borders and Background of Paragraphs 39
- Styles and Style Templates 40
- Hidden Text 42

Basic Operations

- Building RichView Document 43
- Scrolling in RichView 46
- Selecting Part of RichView Document 47
- RichView Clipboard Functions Overview 48
- Searching and Replacing in RichView and RichViewEdit 49
- Obtaining Items of RichView 50
- Modifying RichView Items 51

Editing

- RichViewEdit: Inserting items in position of caret 52
- Undo/redo in RichViewEdit 54
- Drag&drop 55

Graphics

- Animated Images 56
- Smooth Image Scaling 57
- Semitransparent Objects 57

Saving and Loading

- RichView Methods for Saving and Loading 58
- RichView Format (RVF) Overview 59
- Export of RichView Contents to HTML 62

Internationalization

- Unicode in RichView 64
- Bi-directional text in RichView 69

Spelling Check

- Live spelling check 69

Important Technical Information

- Viewer vs Editor 71
- Controls, Documents, Items 72
- Valid Documents 74
- Units of Measurement 74
- Custom Drawing 76
- TRichView Cursors 77
- RichView Format (RVF) Specification 77

1.1.1 Parts of Documents

1.1.1.1 TRichView Item Types

Document in TRichView can contain items of the following types:

- **Text** 88 can have different fonts. Each text item is linked to one item in the collection of text attributes ("styles") 404. A text style defines a font of the text item and some additional effects (such as background color, vertical offset (for subscripts/superscripts), overline), it has some properties affecting editing. A text style can define a hyperlink; for hyperlinks you can define additional properties such as mouse cursor and hover color. A text style also defines whether the text is in ANSI or Unicode 64 encoding.
- **Tabulators** 89 are used instead of TAB (#9) characters.
- **Pictures** 90: you can insert any Delphi graphic type in RichView. You can use TBitmap, TIcon, TMetafile, TJpegImage or third-party graphic classes 671. Gif images can be animated.
- **"Hot Pictures"** 93: the same as pictures, but they are also hypertext links.
- **"Bullets"** 96 are pictures from ImageList. Bullets provide an easy and efficient way for adding many similar "standard" pictures in the document. You can use any number of ImageLists. These bullets have no relation to paragraph bullets and numbering; see "list markers" below.
- **"Hotspots"** 97 are bullets-"hypertext links". They can specify two pictures: normal and "hot" (for highlighting effect under the mouse pointer)
- **"Breaks"** 93 are horizontal lines.
- **Controls** 94: You can use any Delphi/C++Builder control merged in text.
- **List Markers** 99 represent paragraph bullets & numbering.
- **Tables** 98 arrange subitems in rows and columns.
- **Labels** 101 are non-text items looking like a text.
- **Numbered sequences** 102 allow to insert numbered fields (like "Fig. 1", "Fig. 2", ...) in text.
- **Endnotes** 103.
- **Footnotes** 104.
- **Sidenotes** 105 are notes displayed in floating boxes.
- **Text boxes** 107
- **References to parent note** 108
- **"Page number" fields** 109
- **"Page count" fields** 110

Non-text items have an associated string referred in this help as a "name of item". RichView does not use these names itself, use them for your own purposes.

Items are organized in paragraphs. You can define your paragraph attributes ("styles") 398 with different left, right and first line indents, with left, right, center or left-right alignment.

All information above can be saved and loaded to stream, file, database field or clipboard in a special format (RVF 59). You can also export document as HTML, RTF file or print it (see export to html 62).

Internally, all item types are implemented as classes. Standard item types are implemented in unit RVItem.

Detailed description of item types is beyond the scope of this help file.

All item types are classes derived from one base class - TCustomRVItemInfo 526.

See also: **detailed overview of item types** 86

1.1.1.2 "Checkpoints"

Checkpoints are...

Each item can have an associated "checkpoint" – an invisible label that you can use for jumping to the specified part of document. Other word-processing tools call similar objects "bookmarks" or "anchors".

Checkpoint has:

- name (any string; RichView does not use it itself);
- *tag* (just like any RichView items tags [34])
- *RaiseEvent* flag; if *RaiseEvent*=True then RichView generates OnCheckpointVisible [238] event when this checkpoint appears in the field of view (usually as a result of scrolling); this feature only works for RichView, and does not work for RichViewEdit (in this version).

Limitations:

- checkpoint names and tags must not contain #0, #1 and #2 characters; failing to follow this rule will cause problems in saving documents in RichView Format [59];
- the following names are reserved: '_footnoteN', '_endnoteN', '_sidenoteN', where N is a number.

Drawing checkpoints

You can set the option to make checkpoints visible: include *rvoShowCheckpoints* in RichView.Options [146]. By default, checkpoints are shown as dotted horizontal lines with a small circle in the top left corner of the associated item. Displaying checkpoints is useful in editing mode.

Checkpoints colors:

- RichView.Style.CheckpointColor [387] for "normal" checkpoints (*clGreen* by default)
- RichView.Style.CheckpointEvColor [387] for checkpoints with *RaiseEvent=True* (*clLime* by default)

You can create your own procedure for drawing checkpoints, see TRVStyle.OnDrawCheckpoint [411].

Using checkpoints

The usual using of checkpoint is obtaining its vertical coordinate (relative to the top of document area) and scrolling document so that the item associated with the checkpoint becomes visible. There are some more interesting applications for checkpoints.

Adding checkpoints in the document

You can add checkpoint at the end of the document using AddNamedCheckpointEx, AddNamedCheckpointExTag, AddNamedCheckpoint, AddCheckpoint, AddCheckpointTag [166]. All these methods have a subset of functionality of the AddNamedCheckpointExTag method.

```
function TRichView.AddNamedCheckpointExTag(const CpName: String;
  RaiseEvent: Boolean; const Tag: TRVTag [645]): Integer;
```

The parameters of this function were described above. This function (and all these functions) returns the index of the added checkpoint in RichView (the first checkpoint has index=0).

These calls are equivalent:

```
AddNamedCheckpointEx(Name, RaiseEvent)
```

and

```
AddNamedCheckpointExTag(Name, RaiseEvent,'');
```

```
AddNamedCheckpoint(Name)
```

and

```
AddNamedCheckpointExTag(Name, False, '');
```

```
AddCheckpointTag(Tag)
```

and

```
AddNamedCheckpointExTag('', False, Tag);
```

```
AddCheckpoint
```

and

```
AddNamedCheckpointExTag('', False, '');
```

Recommended for using: AddCheckpoint, AddNamedCheckpointEx, AddNamedCheckpointExTag.

As it was told, the above described methods add a new checkpoint to the end of the document. When you add a new item to the end of the document, the last added checkpoint becomes associated with this item. (So, TRichView can contain any number of checkpoints associated with items and only one checkpoint at the end of the document).

You can modify the checkpoint by SetCheckpointInfo 227 method, and remove it by RemoveCheckpoint 214.

The rest of methods can be separated into two groups:

1. Methods returning information about some checkpoint. These methods return a value of TCheckpointData class.
2. Methods extracting checkpoint properties from the TCheckpointData value.

Methods and events of the first group (obtaining TCheckpointData):

TCheckpointData is a pointer. If these methods return **nil**, it means that there is no such checkpoint, item has no associated checkpoint, etc.

- GetFirstCheckpoint 188, GetNextCheckpoint 199(CheckpointData), GetLastCheckpoint 198, GetPrevCheckpoint 202(CheckpointData) – allow you to obtain a list of all checkpoints in the document (except for checkpoints in table 111 cells, see below);
- GetItemCheckpoint 191(ItemNo) – returns the checkpoint associated with the specified item;
- GetCheckpointByNo 185(No) – returns the checkpoint with the specified index;
- FindCheckpointByName 182(Name) – returns the first checkpoint having the specified Name;
- FindCheckpointByTag 182(Tag) – returns the first checkpoint having the specified Tag;
- event OnCheckpointVisible 238 (Sender: TRichView; CheckpointData: TCheckpointData); this event is generated when:
  - RichView.CPEventKind 139 = *cpeWhenVisible* and some checkpoint (with RaiseEvent=*True*) becomes visible *(example of application: you can mark pictures in TRichView with checkpoints; when this picture becomes visible you can display information about it in a separate window)*
  - RichView.CPEventKind = *cpeAsSectionStart*; in this mode it's guaranteed that the last event is generated for visible checkpoint or for the nearest checkpoint above the visible area *(example of application: you can mark beginnings of chapters of your document with checkpoints, create a table of contents in ListBox (one listbox item = one chapter = one "RaiseEvent" checkpoint); if you select listbox item associated with the last raised checkpoint, the list box will always have name of the chapter in the field of view highlighted)*

Methods of the second group (extracting information from TCheckpointData):

- GetCheckpointInfo 185(CheckpointData, Tag, Name, RaiseEvent) – returns Tag, Name and "RaiseEvent" flag of checkpoint;
- GetCheckpointXY 186(CheckpointData, X,Y), GetCheckpointYEx 187(CheckpointData): Integer – obtaining information about coordinates of checkpoint (Y is the most useful; you can pass this Y to RichView.ScrollTo 513 method);
- GetCheckpointItemNo 186(CheckpointData) : Integer – returns the index of the associated item; it can return -1 for checkpoint in the end of document (no associated item);
- GetCheckpointNo 186(CheckpointData): Integer – returns the index of the checkpoint;

Methods belonging to the both groups:

- GetCheckpointY 186(no): Integer – equivalent to GetCheckpointYEx(GetCheckpointByNo(No));

Methods of editor

The first group of methods contains editing-style analogs of SetCheckpointInfo and RemoveCheckpoint:

- SetCheckpointInfoEd 340 – adds checkpoint for the specified item, as an editing operation;
- RemoveCheckpointEd 333 – deletes checkpoint for the specified item, as an editing operation.

The second group contains methods working with the item at the position of caret (if the caret is between items, they work with the item to the left):

- GetCurrentCheckpoint 309 – returns checkpoint associated with the item at the caret position;
- SetCurrentCheckpointInfo 343 – adds/modifies checkpoint for the item at the caret position, as an editing operation;
- RemoveCurrentCheckpoint 334 – deletes checkpoint for the item at the caret position, as an editing operation.

The third group contains methods working with the checkpoint at the position of caret:

- InsertCheckpoint 318 – inserts checkpoint in the caret position (adds/modifies checkpoint for the item to the right; if the caret was in the middle of text item, the method preliminarily splits this item at the caret position), as an editing operation;
- GetCheckpointAtCaret 307 – returns the checkpoint exactly at the caret position;
- RemoveCheckpointAtCaret 333 – deletes the checkpoint exactly at the caret position, as an editing operation.

RTF and HTML

In HTML 62, checkpoints are saved as anchors (<a name> tag). Either checkpoint name or checkpoint index can be used, see *rvsoUseCheckpointsNames* option 639 for SaveHTML 216 and SaveHTMLEx 217. Links to checkpoints must start with '#' character.

In RTF, checkpoints are saved as bookmarks (checkpoints names are used). When importing RTF, bookmark start is read as checkpoint (bookmark name is written in checkpoint name). When reading RTF, links to bookmarks are converted to links started from '#'. When writing RTF, links started from '#' are written as links to bookmarks.

Example 1

Enumerating all checkpoints in the document (not including checkpoints in table cells). This method is very fast even for large documents, because all checkpoints are organized in a special list.

```
var Tag: TRVTag 645;
    Name: String;
    RaiseEvent: Boolean;
    CheckpointData: TCheckpointData;
begin
  CheckpointData := MyRichView.GetFirstCheckPoint 188;
  while CheckpointData<>nil do begin
    MyRichView.GetCheckpointInfo 185(CheckpointData,Tag,Name,RaiseEvent);
      // processing this checkpoint
    ...
    CheckpointData := MyRichView.GetNextCheckpoint 199(CheckpointData);
  end;
end;
```

Example 2

Enumerating all checkpoints in the document, including checkpoints in table cells

```
procedure EnumCheckpoints(RVData: TCustomRVData 608);
var i,r,c: Integer;
  table: TRVTableItemInfo 528;
  Tag: TRVTag 645;
  Name: String;
```

```
  RaiseEvent: Boolean;
  CheckpointData: TCheckpointData;
begin
  for i := 0 to RVData.ItemsCount-1 do begin
    CheckpointData := RVData.GetItemCheckpoint[191](i);
    if CheckpointData<>nil then begin
      RVData.GetCheckpointInfo[185](CheckpointData,Tag,Name,RaiseEvent);
      // processing this checkpoint
      ...
    end;
    if RVData.GetItemStyle[194](i)=rvsTable then begin
      table := TRVTableItemInfo[528](RVData.GetItem[190](i));
      for r := 0 to table.RowCount-1 do
        for c := 0 to table.ColCount-1 do
          if table.Cells[537][r,c]<>nil then
            EnumCheckpoints(table.Cells[537][r,c].GetRVData[575]);
    end;
  end;
end;
```

Call:

```
EnumCheckpoints(MyRichView.RVData[151]);
```

In the second example, RVData.GetCheckpointXY[186] returns checkpoint coordinates relative to the top left corner of this RVData (it may be left top corner of the cell). In order to get absolute coordinates of this checkpoint in the document (for MyRichView.ScrollTo[513]), obtain the coordinates of top left corner of this RVData (TCustomRVFormattedData(RVData).GetOriginEx(X,Y)), relative coordinates of checkpoint (TCustomRVFormattedData(RVData).GetCheckpointXY(X,Y)), and sum them up

Compatibility Issues

Incompatibility with old (freeware) versions of RichView: in older versions checkpoints were items. Now it's not true – they are an additional information associated with items (text, pictures or controls). Therefore, you now can't add two or more checkpoints one by one without an item "between" them. You can choose one of two options when compiling RichView source:

- RichView raises an exception when you try to add checkpoint after checkpoint (default mode);
- RichView ignores the second checkpoint; Add***Checkpoint*** methods will return the index of previously added checkpoint in this case. This mode is turned on by defining RVALLOWCPBYCP conditional define.

See also...

See also properties and events of TRVStyle:

CheckpointColor, CheckpointEvColor[387], OnDrawCheckpoint[411]

See also properties of TRichView:

Options[146],.CPEventKind[139]

See also events of TRichView:

OnCheckpointVisible[238]

See also methods of TRichView:

AddNamedCheckpointEx, AddNamedCheckpointExTag, AddNamedCheckpoint, AddCheckpoint, AddCheckpointTag[166], SetCheckpointInfo[227], RemoveCheckpoint[214],·GetFirstCheckpoint[188], GetNextCheckpoint[199], GetItemCheckpoint[191], GetCheckpointByNo[185], FindCheckpointByName[182], FindCheckpointByTag[182], GetCheckpointInfo[185], GetCheckpointXY[186], GetCheckpointYEx[187], GetCheckpointItemNo[186], GetCheckpointNo[186], GetCheckpointY[186]

See also methods of TRichViewEdit:

GetCurrentCheckpoint 309, SetCheckpointInfoEd 340, RemoveCheckpointEd 333

1.1.1.3 Items' "Tags"

Each item of RichView has an associated string value – *tag*. *Tags* are provided for the convenience of storing additional string value for special needs in your application. The main purpose of tags is organizing a work with a hypertext.

Do not confuse RichView *items tags* with "Tag" properties of TComponent descendants! (RichView uses "Tag" properties of inserted controls for its own needs; you can set "Tag" properties of ImageList for organizing saving/loading "bullets" and "hotspots" from RVF; but this topic is about completely different tags!)

What you need to know about items tags:

- if you do not want to use them – just forget about them; they always will be equal to '';
- "checkpoints" 30 also have tags; all rules for items tags work for checkpoints tags;
- table cells also have tags 574; all rules for items tags work for checkpoints tags;
- editor can clear (set to '') tags when applying different style to text according to RichViewEdit.EditorOptions 291;
- If you want to save and load document in RVF you must provide that tag strings do not contain #1 and #2 characters;
- You can use AddXXX methods 43 which do not have Tag parameter. These functions create new items with Tag=''.

Unicode note: For Delphi/C++Builder 2009 or newer, tags are Unicode strings. For older versions of Delphi, tags are ANSI strings.

Example:

```
var Tag: TRVTag;
    S: String;
// Converting the first text item and its tag string to upper case
  MyRichView.GetTextInfo(0, S, Tag);
  MyRichView.SetItemText(0, UpperCase(S));
  MyRichView.SetItemTag(0, UpperCase(Tag));
```

Tags in older version of TRichView

In TRichView versions prior to v13.2, tags were Integers. Depending on *rvoTagsArePChars* Option 146, they can be either plain integer values or pointers to strings.

This mode can be returned by defining RVOLDTAGS in RV_Defs.inc.

See also...

RichView methods for adding items with tags to the end of document:

AddNLTag, AddTag 167, AddBreakTag 160, AddPictureExTag 169, AddHotspotExTag 164, AddBulletExTag 160, AddControlExTag 162, AddBreakExTag 160.

RichView methods for adding "checkpoints" with tags to the end of document:

AddCheckpointTag, AddNamedCheckpointExTag 166.

RichView methods for obtaining information about items (including tags):

GetItemTag 195, GetBreakInfo 183, GetBulletInfo 184, GetHotspotInfo 189, GetPictureInfo 201, GetControlInfo 187, GetTextInfo 204.

RichView methods for modifying items (including tags):

SetItemTag 231, SetBreakInfo 226, SetBulletInfo 227, SetHotspotInfo 229, SetPictureInfo 234, SetControlInfo 228.

RichView methods allowing work with checkpoints tags:

FindCheckpointByTag 182, GetCheckpointInfo 185, SetCheckpointInfo 227.

RichViewEdit methods for obtaining information about item in caret position:

GetCurrentTag 316, GetCurrentBreakInfo 308, GetCurrentBulletInfo 308, GetCurrentHotspotInfo 310, GetCurrentPictureInfo 315, GetCurrentControlInfo 309, GetCurrentTextInfo 316.

RichViewEdit methods for inserting:

InsertStringTag, InsertStringATag, InsertStringWTag 326

RichViewEdit methods for modifying items (including tags):

SetItemTagEd 352, SetBreakInfoEd 339, SetBulletInfoEd 339, SetHotspotInfoEd 349, SetPictureInfoEd 354, SetControlInfoEd 341.

RichViewEdit methods for modifying information about item in caret position:

SetCurrentTag 349, SetCurrentBreakInfo 342, SetCurrentBulletInfo 342, SetCurrentHotspotInfo 345, SetCurrentPictureInfo 348, SetCurrentControlInfo 344

RichViewEdit methods for working with checkpoints tags:

SetCheckpointInfoEd 340, SetCurrentCheckpointInfo 343.

RichViewEdit options affecting tags:

EditorOptions 291.

1.1.1.4 Hypertext Overview

TRichView has no default actions on hypertext events. However, it provides a convenient way for you to create hypertext document and define any actions that you want in response on user click on documents. RichView provides visual effects for items-hypertext links (highlighting).

There are the following types of hypertext items:

- text of hypertext styles 445;
- tabs 89 and label items 101, linked to hypertext styles 445;
- *hot pictures* 93
- *hotspots* 97

RichView provides the following indication of hypertext items (when the mouse pointer moves above them):

- changing color of text items (TFontInfo.HoverColor 436, HoverBackColor 436);
- underlining text items (TFontInfo.HoverEffects 437);
- changing mouse cursor (any hypertext text style can have its own cursor, see TRVStyle.JumpCursor 394, TFontInfo.JumpCursor 438)
- for hotspots 97 it can change picture (there are two pictures for hotspot – "normal" and "hot"; they must be in the same imagelist but can have different indices);

By default, main events of hypertext (OnRVMouseMove 254, OnJump 247) and cursor indication do not work in editor (TRichViewEdit 282). Instead of highlighting items under mouse pointer editor highlights the item at the position of caret (item that is being edited).

But editor can be switched to hypertext mode when user presses and holds Ctrl key, if *rvoCtrlJumps* is in EditorOptions 291.

Since version 1.6, hypertext works in ReadOnly 294 editor just like in viewer (since version 13, this behavior can be turned off using *rvoNoReadOnlyJumps* option from EditorOptions 291)

Besides, you can provide some specific actions in response on user clicking any RichView item, using OnRVMouseDown 253 and OnRVMouseUp 254 events.

Hypertext

Any hypertext link (text, *hotspot* or *hot picture*) has its own **id**. This **id** is passed in hypertext events - OnRVMouseMove[254] and OnJump[247]. The first hypertext item has **id** equal to RichView.FirstJumpNo[144], the next one is greater by one and so on. By default, RichView.FirstJumpNo=0 and hypertext items have **id** = 0, 1, 2, ...

Use OnRVMouseMove[254] to provide action when user moves cursor to hypertext link. For example, you can change text in the status bar.

Example

```
procedure TMyForm.MyRichViewRVMouseMove(Sender: TObject; id: Integer);
begin
  if id=-1 then
    MyStatusbar.SimpleText := ''
  else
    MyStatusbar.SimpleText := 'Click here';
end;
```

Special value (-1) is passed to this event as **id** when the user moves the mouse pointer outside any hypertext link. This event never occurs with the same **id** two times one after one.

The main hypertext event is OnJump[247].

Example

```
procedure TMyForm.MyRichViewJump(Sender: TObject; id: Integer);
begin
  case id of
    0: Application.MessageBox('Ah!','',0);
    1: Application.MessageBox('Ugh!','',0);
    2: Application.MessageBox('Oops!','',0);
  end;
end;
```

The method shown in the example above (using **id** for defining specific actions) was the only available method in the older versions of RichView. Now you can use other ways, described below.

Sometimes information contained in the clicked item is enough to know what to do. The key method here is GetJumpPointLocation[197] This function allows you to obtain the index of the clicked item. For example, if you have only text links (no *hotspots* and *hot pictures*) and they are URLs, you can write:

Example

```
uses ..., CRVFData, ShellApi;
...
procedure TMyForm.MyRichViewJump(Sender: TObject; id: Integer);
var URL: String;
    RVData: TCustomRVFormattedData[608];
    ItemNo: Integer;
begin
  MyRichView.GetJumpPointLocation[197](id, RVData, ItemNo);
  URL := RVData.GetItemText[195](ItemNo);
  ShellExecute(Application.Handle, 'open', PChar(URL),
    nil, nil, SW_NORMAL);
end;
```

In the example above, we use GetJumpPointLocation[197] to get the document containing this hyperlink (RVData) and the index of the clicked hypertext item in this document (ItemNo).

Next, we get information about the clicked text item with GetItemText and use the received string to launch a

browser.

If you have both text and *hotspot/hot pictures* hypertext items, you can use RVData.GetItemStyle[194] method to recognize the item type ("style"), and next use GetTextInfo[204] or GetHotspotInfo[189]/GetPictureInfo[201]. For *hotspots* you can use image indices and *hotspot* name to decide which action to do.

The method shown in the next example is universal. You can use *tags* to add your information to any RichView item. (See tags[34]).

Example

```
uses ..., CRVFData, ShellApi;
...
procedure TMyForm.MyRichViewJump(Sender: TObject; id: Integer);
var URL: String;
    RVData: TCustomRVFormattedData[608];
    ItemNo: Integer;
begin
  MyRichView.GetJumpPointLocation[197](id, RVData, ItemNo);
  URL := RVData.GetItemTag[195](ItemNo);
  ShellExecute(Application.Handle, 'open', PChar(URL),
    nil, nil, SW_NORMAL);
end;
```

Inserted controls cannot be hyperlinks. But, of course, you can set your own cursors and mouse events for any inserted control.

Hypertext and tables

Hypertext works not only in a the main document but also in table cells (subdocuments)

Hypertext links are numbered sequentially through the document, including links in cells. All links in the document have unique indices (**id**).

If you have a hypertext **id** you can obtain the hypertext item location with the method:

```
procedure TRichView.GetJumpPointLocation[197](id: Integer;
  var RVData: TCustomRVFormattedData; var ItemNo: Integer);
```

A hypertext item is the ItemNo-th item in RVData object.

RVData can be RichView[Edit].RVData[151], cell[567], or RVData[151] of inplace editor.

Method GetJumpPointItemNo is obsolete because it returns only ItemNo.

See also...

- OnReadHyperlink[248], OnWriteHyperlink[267];
- example how to create a list of hyperlinks and move the caret to the selected hyperlink: ***http://www.trichview.com/forums/viewtopic.php?t=397***.

1.1.1.5 Paragraphs

Paragraphs

TRichView documents are organized in paragraphs. Each paragraph has its attributes ("style") defined as an index in the collection of paragraph styles (RichView.Style[156].ParaStyles[398]).

For the list of properties available for paragraphs, see TParaInfo[448] class (TParaInfo is a class of item in the collection of paragraph styles).

Line breaks inside paragraphs

(Since version 1.2) You can move am item to the new line within the same paragraph (press Shift+Enter instead of Enter). Visually, such line breaks have the following differences from paragraph breaks:

- FirstIndent[451] is not applied to the first line after the break;
- no additional spacing between this line and the previous line (SpaceBefore[455] and SpaceAfter[454]) is applied;
- if the paragraph has border/background[39], this line will be inside the same border/background as the previous line.

Specifying paragraph attributes when generating document

All methods of RichView for appending items (except for the obsolete methods) have **ParaNo** parameter. It can be:

- *zero or positive value*: index in the collection of paragraph styles, if this item will start a new paragraph;
- *-1*, if this item will continue the paragraph (*please do not add items with ParaNo=-1 after breaks*[93] or tables[528]).

Breaks (horizontal lines) do not have paragraph style, so AddBreak*** methods do not have ParaNo parameter. But SpaceBefore[455] and SpaceAfter[454] values of the 0-th paragraph style are added to spacing of all breaks.

These methods are:

```
procedure AddNL(s, StyleNo, ParaNo);
procedure AddFmt(FormatStr, Args, StyleNo, ParaNo);
procedure AddTextNL(s, StyleNo, FirstParaNo, OtherParaNo);
procedure AddTextNLA(s, StyleNo, FirstParaNo, OtherParaNo);
procedure AddTextNLW(s: WideString; StyleNo, FirstParaNo, OtherParaNo,
  DefAsSingleParagraph);
procedure AddPictureEx(Name, gr, ParaNo, VAlign);
procedure AddHotspotEx(Name, ImageIndex, HotImageIndex,
  ImageList, ParaNo);
procedure AddBulletEx(Name, ImageIndex, ImageList, ParaNo);
procedure AddControlEx(Name, ctrl, ParaNo, VAlign);
procedure Add(s, StyleNo); // equivalent to AddNL(s, StyleNo, -1).
```

There are some obsolete methods that were developed when RichView had no paragraph styles. They still work, see compatibility[705].

See also: building a document[43], item types[29].

All the methods above have "tagged" analogs, see "tags"[34].

See also methods of RichView:

```
function LoadText(FileName, StyleNo, ParaNo, AsSingleParagraph):Boolean;
function LoadTextW(FileName, StyleNo, ParaNo, DefAsSingleParagraph):Boolean;
function AppendRVFFromStream(Stream, ParaNo):Boolean;
function IsParaStart(ItemNo): Boolean;
function IsFromNewLine(ItemNo): Boolean;
function GetItemPara(ItemNo): Integer;
```

Text flow around left- and right-aligned items

Some items (for example, pictures[90]) can be aligned to the left or the right side (see TRVVAlign[648]). For such items, an invisible placeholder is left in the place of their insertion; the item itself is placed at the specified side. If the item is at the beginning of the line, this item is placed on the same level; otherwise, it is placed below this line.

Paragraphs flow around side-aligned images. You can disallow flowing around left- and/or right-aligned items using ClearLeft[138] and/or ClearRight[138] properties.

Editing

RichViewEdit has a group of Insert*** methods inserting new item(s) in the position of caret, using the current paragraph style. Usually, if you implement RichViewEdit-based editor, you need to provide interface for displaying and changing the current paragraph style. You can do it using ComboBox with style names, toolbar buttons or/and keyboard shortcuts.

```
property CurParaStyleNo[289]: Integer;
```

– index of the current (in the position of caret) paragraph style;

When the user moves the caret to paragraph with different style, OnCurParaStyleChanged[358] event is generated, so you can select another item in ComboBox of styles or check toolbar buttons.

Assigning to CurParaStyleNo property is possible, but will not create a desired effect. To apply style to the paragraph at the position of the caret (and all selected paragraphs) use

```
procedure ApplyParaStyle[297](ParaStyleNo: Integer);
```

ApplyParaStyle applies only one paragraph style to all selected paragraphs. This method does not allow to implement more complex commands, such as "increase indents", "align to the right", etc. For implementing these commands, use

```
procedure ApplyParaStyleConversion[298](UserData: Integer);
```

See also...

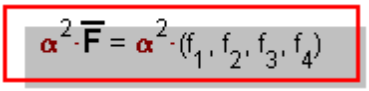
Borders and Background of Paragraphs[39]

1.1.1.6 Borders and Background of Paragraphs

Paragraphs can have borders and/or background color. These properties are defined in paragraph attributes ("styles") (see TParaInfo[448]). Border is defined in Border[451] property, background is in Background[450] property.

By default, there is no border around paragraph (TRVStyle.ParaStyles[i].Border.Style[468]=*rvbsNone*), and background of paragraph is transparent (TRVStyle.ParaStyles[i].Background.Color=*clNone*).

Border offsets and colored area offsets are defined separately from each other, so you can draw border inside colored area, border around colored area with gap, and even more interesting effects:



Example of paragraph border and background

Border and colored area both have BorderOffsets property, which defines:

- for borders: a gap between the border and the paragraph contents;
- for background: a colored area overhang (padding) from the paragraph content.

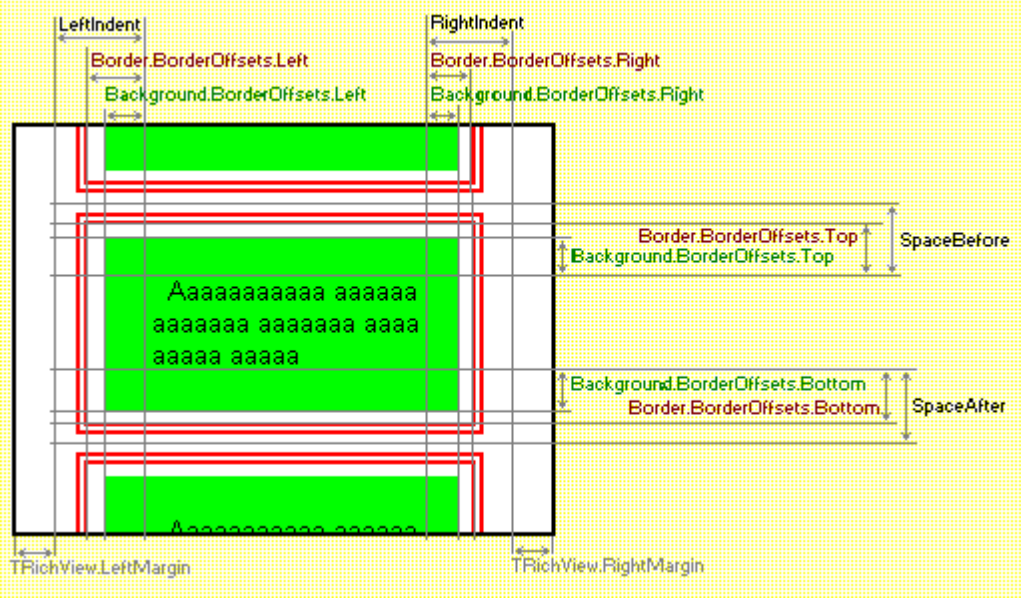
Top, right, bottom, left offsets (i.e. gap and padding) are defined by Top, Right, Bottom, Left properties of BorderOffsets.

Positive values increase gap (padding). Negative values put the border and edges of the colored area inside the paragraph content.

In the example above, Background.BorderOffsets = (Left: -7; Top: -7; Right: 7; Bottom: 7), and values are measured in pixels.

BorderOffsets.Top must not exceed SpaceBefore[455] property of the same paragraph.

BorderOffsets.Bottom must not exceed SpaceAfter[454] property of the same paragraph.



Scheme: properties of paragraph border, background and layout

You can hide some edges of border using Border.VisibleBorders[468] property.

Breaks[93] use SpaceBefore and SpaceAfter values of the 0-th paragraph style, but they never have a border and a colored background.

It's possible to create a line break without starting a new paragraph. In editor, press Shift+Enter instead of Enter. Items after the caret will be moved to the new line, but still they will be inside the same paragraph (inside the common border and the colored area, no SpaceBefore and SpaceAfter will be applied).

See also...

TParaInfo.Border[451]

TRVBorder[467]

1.1.1.7 Styles and StyleTemplates

As you could see, this manual sometimes calls items in TextStyles[404] and ParaStyles[398] "styles", and sometimes it calls them simply "text/paragraph attributes". Apart from historical reasons, there are other reasons for this dual naming, resulting from different cases of use.

Using a predefined set of styles

In this mode, you have some predefined set of items in TextStyles[404] and ParaStyles[398], and they are not changed while the application works.

In this mode, these items may be called "styles".

Example

In this example, RichView1: TRichView[128] is linked to RVStyle1: TRVStyle[385].

```
RVStyle1.TextStyles.Clear;
with RVStyle1.TextStyles.Add do begin
  StyleName[432] := 'Normal Text';
end;
with RVStyle1.TextStyles.Add do begin
  StyleName := 'Title Text';
  Size[441] := 14;
  Style[441] := [fsBold];
```

```
end;
RVStyle1.ParaStyles.Clear;
with RVStyle1.ParaStyle.Add do begin
  StyleName := 'Normal Paragraph';
end;
with RVStyle1.ParaStyle.Add do begin
  StyleName := 'Title Paragraph';
  Alignment[449] := rvaCenter;
  Options[453] := [rvpaoKeepWithNext];
end;
...
RichView1.Clear[175];
RichView1.AddNL[167]('Chapter 1', 1, 1);
RichView1.AddNL('Once upon a time...', 1, 1);
RichView1.Format[182];
```

You can change properties of styles to change the appearance of the created document, without recreating the document itself:

```
with RVStyle1.TextStyles[1] do begin
  Color[435] := clRed;
  FontName[436] := 'Verdana';
end;
with RVStyle1.ParaStyles[1] do begin
  Alignment := rvaLeft;
end;
RichView1.Reformat[214];
```

You can configure[134] RichView1 to allow insertion of RVF and RTF files/streams without adding new styles.

However, this case of usage is very limited. Normally, users want to apply new formatting to parts of a document: change colors, fonts, alignments, etc., so it is impossible to create all necessary styles initially.

Using a dynamic set of styles

In this mode, you can have some initial set of items in TextStyles[404] and ParaStyles[398]; but the most of items are added while the application works. For example, new items may be added when inserting RVF or RTF files/streams, or when applying changes to the selection (ApplyStyleConversion[299], ApplyParaStyleConversion[298]). RichViewActions add new items as well.

In this mode, items of TextStyles[404] and ParaStyles[398] work more like direct text/paragraph attributes rather than styles: the most of them do not have a special unique name or meaning.

Some items of TextStyles[404] and ParaStyles[398] may be marked as Standard[431] (usually, the items added initially). They may have a special meaning, they are not removed by DeleteUnusedStyles[181], they can be saved in RTF as a style sheet (if *rvrtfSaveStyleSheet* in RTFOptions[150]). However, this is not a complete solution for styles, because all other items are completely independent of them: if you change properties of standard styles, only text/paragraphs formatted using them are changed; all other parts of the document are unstyled. It makes standard styles almost useless, so usually an application should create only one standard text style and one standard paragraph style (defaults for new documents).

The complete solution for implementing real text and paragraph styles is "style templates".

Style templates (+dynamic set of styles)

StyleTemplates[402] is a collection in TRVStyle[385] component. Unlike TextStyles[404] and ParaStyles[398], style templates do not define text and paragraph properties of document items[85] and paragraphs directly. Instead, they provide a mechanism for modifying TextStyles[404] and ParaStyles[398].

Using style templates is optional; they are used only if TRichView.UseStyleTemplates[158]=*True*.

Document items do not have direct links to style templates, they are still linked with TextStyles[404] and

ParaStyles[398]. But items in TextStyles[404] and ParaStyles[398] may be linked to style templates using StyleTemplateId[432] property.

Each item in a document may be (indirectly) linked to up to 2 style templates:

- via its paragraph style, or
- via its text style (if this is a text item[88], tab[89], label[101], sequence[102], footnote[104], endnote[103], or note reference[108]).

It's recommended (but not necessary) to link every paragraph style to some style template. The recommended name[461] for the default paragraph style template is "Normal".

As for text styles, it is recommended to link them to style templates only if it is necessary.

There are 3 kinds[461] of style templates: applicable to text, applicable to paragraphs, applicable to text and paragraphs.

Each style template may have properties of both text[463] and paragraph[463]. Unlike items of TextStyles or ParaStyles, a style template:

- may define only a subset of text and paragraph properties (listed in ValidTextProperties[464] and ValidParaProperties[463]);
- may inherit text and paragraph properties from a parent style template[463].

When some document item is (indirectly) linked to both a paragraph style template and a text style template, both these style templates are linked with its text style: a text style template as TextStyles[404][StyleNo].StyleTemplateId[432], a paragraph style template as TextStyles[404][StyleNo].ParaStyleTemplateId[439]. TRichView automatically provides for each document item that TextStyles[404][StyleNo].ParaStyleTemplateId[439] = ParaStyles[398][ParaNo].StyleTemplateId[432]. In such cases, a text style template has a priority in defining text attributes: a paragraph style template defines only text attributes not defined by a text style template.

Not all properties of items in TextStyles[404] and ParaStyles[398] are defined by their style template: they can have some properties changed. A list of changed properties is not stored, it is calculated when it is needed. These changes may be (optionally) kept when applying another style template to these items, and they are kept when StyleTemplates[402] are edited.

TRichViewEdit has the following methods for applying the specified style template to the selection: ApplyStyleTemplate[299], ApplyTextStyleTemplate[301], ApplyParaStyleTemplate[298]. The same methods may be used to clear a style template or additional formatting.

Style templates can be edited using ChangeStyleTemplates[304] method.

1.1.1.8 Hidden Text

Hidden items

Some items can be hidden. Hidden items are displayed only if *rvoShowHiddenText* is included in the Options[146] property of TRichView[128] control (they are displayed with a special dotted underline).

Text items[88] are hidden, if *rvteoHidden* is included in the Options[438] property of their style[444].

Non-text items[29] are hidden, if a non-zero value is assigned to *rvepHidden* extra item integer property[627].

Import and Export

RVF (RichView Format): hidden items are saved and loaded as hidden items.

RTF (Rich Text Format): hidden items are saved as hidden text and objects. Loading depends on the value of RTFReadProperties[151].SkipHiddenText[273] property of TRichView[128] control. By default (*False*), hidden text and objects are not loaded from RTF (skipped).

HTML: The CSS version of HTML export (SaveHTMLEx[217]) saves hidden items as hidden text and objects. The simple version (SaveHTML[216]) saves hidden items like normal items.

Text: the text saving methods save a hidden text like a normal text. However, when copying selection to the

Clipboard as a text, hidden text is not included.

Performance

A text formatting procedure is somewhat faster, if the option for displaying hidden items (*rvoShowHiddenText* in Options[146]) is set.

1.1.2 Basic Operations

1.1.2.1 Building a Document

Creating a New Document

You can build document by:

1. clearing TRichView,
2. appending items,
3. reformatting and redrawing RichView.

Therefore, a general structure of document generation procedure is:

```
MyRichView.Clear[175];
<Methods for adding contents>
MyRichView.Format[182];
```

Note: all methods introduced in RichView (except for the formatting methods) do not reformat document, do not redraw RichView and do not call OnChange[357] event (RichView does not have OnChange event, it is introduced in TRichViewEdit) (viewer-style methods)

A group of methods for adding items is the largest group of methods (and contains a large number of methods for backward compatibility).

Appending Text Items[88]

Important: Do not pass string with cr, lf or cr+lf, tab and page break characters in the functions below, except for AddText*** methods!

Some text items can be hypertext. Hypertext items are added by the same methods as regular text items.

Methods working with String parameter

These methods add text specified in a String parameter. This parameter is Unicode for Delphi/C++Builder 2009, and ANSI for the older versions of Delphi/C++Builder.

- AddNL(S, StyleNo, ParaNo)[167] – adds one item with text *S*; *StyleNo* is an index in the collection of text styles (RichView.Style.TextStyles[404]); *ParaNo* is an index in the collection of paragraph styles (RichView.Style.ParaStyles[398]) (if this item will start a new paragraph) or -1 (if this item will continue the paragraph).
- Add(S, StyleNo)[167] – equivalent to AddNL(S, StyleNo, -1).
- "Tagged" analogs: AddNLTag[167], AddTag[167] (see *Tags*[34])
- AddTextNL(s, StyleNo, FirstParaNo, OtherParaNo)[171] – adds one or more text items; the first item is added with FirstParaNo paragraph style, and other items are added as new paragraphs with OtherParaNo paragraph style; items are separated by cr, lf or cr+lf characters in s; s can contain page break and tab characters.
- AddFmt(FormatStr, Args, StyleNo, ParaNo)[163] adds Format(FormatStr, Args) string, where Format is a function from SysUtils unit.

Methods working with ANSI/Unicode explicitly

- AddNLATag, AddNLWTag[167], AddTextNLA, AddTextNLW[171]

Notes for former users of freeware version:

In older versions of RichView you was not able to add empty strings, which are useful as empty paragraphs. Now there is no such limitation.

In older versions, there was no special processing of tab characters: they were displayed as special characters. Now Add*** methods replace tab characters with spaces or insert them as tabulators 89 (RichView.Style.SpacesInTab 402 space characters per one tab; changing SpacesInTab will not change already created document).

Appending Images 90

You can add picture of any Delphi format – TBitmap, TIcon, TMetafile, TJpegImage or third party graphic format 671.

You should create a picture and TRichView will destroy it itself when necessary (for example, when you call RichView.Clear 175).

- AddPictureEx(Name, gr, ParaNo, VAlign) 169
  (Name: TRVAnsiString 623 – name of picture; this string is not used by RichView itself; this string should not contain cr and lf characters; gr: TGraphic - picture; ParaNo:Integer is an index in the collection of paragraph styles (RichView.Style.ParaStyles) (if this picture will start the paragraph) or -1 (if this picture will continue paragraph); VAlign: TRVVAlign – vertical alignment of picture).
- "Tagged" analogs: AddPictureExTag 169 (See *Tags* 34).
- Obsolete method: AddPicture 169.

Appending *Hot Images* 93

- AddHotPicture, AddHotPictureTag 163

Appending Delphi Controls 94

You can add any Delphi control in RichView, and it will work as usual. You should create the control and RichView will destroy it itself when necessary (for example, when you call RichView.Clear). The methods for adding controls assign RichView to control.Parent.

- AddControlEx(Name, ctrl,ParaNo, VAlign) 162, AddControlExTag(Name, ctrl,ParaNo, VAlign, Tag) 162 is similar to the methods for pictures.
- Obsolete method: AddControl 162.

This version of TRichView, like old ones, uses Tag properties of inserted components for internal needs (do not confuse with RichView items tags).

Appending *Bullets* 96 – Images from ImageLists

- AddBulletEx (Name, ImageIndex, ImageList, ParaNo) 160
  (Name: TRVAnsiString 623 – name of the bullet; this string is not used by RichView itself; this string should not contain cr and lf characters; ImageIndex: Integer – index of the picture in imagelist; ImageList: TCustomImageList – imagelist; this will be just a reference to imagelist; RichView does not hold copy of imagelist, and does not free it; ParaNo:Integer is an index in the collection of paragraph styles (RichView.Style.ParaStyles) (if this bullet will start a paragraph) or -1 (if this bullet will continue paragraph)).
- "Tagged" analog: AddBulletExTag 160 (See *Tags* 34).
- Obsolete method: AddBullet 160.

Appending *Hotspots* 97 – Hypertext Images from ImageLists

Hotspots are just like *bullets*, but can have two images ("normal" and "hot") and can be used as hypertext link.

- AddHotspotEx(Name, ImageIndex, HotImageIndex, ImageList, ParaNo) 164 – similar to AddBulletEx

- "Tagged" analog: AddHotspotExTag[164] (See *Tags*[34]).
- Obsolete method: AddHotspot[164].

Appending *Breaks*[93] – Horizontal Lines

- AddBreak, AddBreakEx, AddBreakExTag, AddBreakTag.[160]

Adding Paragraphs' Bullets and Numbering[99] (list markers)

List markers can be added using SetListMarkerInfo[233] method. Unlike the methods above, this method can insert list markers not only to the end of document, but set them for existing paragraphs.

Adding Other Items

Tables[98], labels[101], numbered sequences[102], endnotes[103], footnotes[104], references to parent footnotes and endnotes[108], custom item types[111] do not have special methods for adding them to documents. All of them are added by AddItem[166] method.

Appending *Checkpoints*

Checkpoints are not items of RichView, but methods for adding *checkpoints* are similar to methods for adding items. The main methods for appending *checkpoints* are AddNamedCheckpointEx, AddNamedCheckpointExTag[166]. See "checkpoints"[30].

Adding Page Breaks

Page break is an item property ("this item starts a new page" flag). Only items that start a new paragraph can start a new page.

Example:

```
var ItemNo: Integer;

MyRichView.AddNLATag('First page', 0, 0, '');
ItemNo := MyRichView.ItemCount;
MyRichView.AddNLATag('Second page', 0, 0, '');
MyRichView.PageBreaksBeforeItems[ItemNo] := True;
```

Appending and Inserting RVF[59]

You can append RVF data by the methods:

- function InsertRVFFromStream[206](Stream: TStream; Index: Integer):Boolean;

This method can insert RVF. The first item of RVF will have the specified Index. For example, if you want to append items from RVF, write:

```
MyRichView.InsertRVFFromStream(MyStream, MyRichView.ItemCount);
```

InsertRVFFromStream is the only method of RichView (not RichViewEdit) for inserting items.

- function AppendRVFFromStream[173](Stream: TStream; ParaNo: Integer):Boolean;

This method is similar to InsertRVFFromStream (if it used for appending RVF), but it ignores paragraph style of the first item in RVF and uses ParaNo instead of it. ParaNo is an index in the collection of paragraph styles (RichView.Style.ParaStyles[398]) (if item will start a paragraph) or -1 (if the item will continue paragraph).

Appending Content of Another RichView

- procedure AppendFrom[172](Source: TCustomRichView);

This is the most efficient method to copy contents of one RichView to another one. But items of some types (including Inserted controls and tables) are not copied with this method (just ignored).

Loading

Methods for loading [58] (**Load*****) belong to this group of methods as well.

Additional Notes

There is an important change in version 1.2: you can add items from new line without starting a new paragraph. There are no new methods for generating documents (it is added in v1.3: AddTextNLW [171]), but there is a method that affects their behavior: TRichView.SetAddParagraphMode [225].

Some custom item types (such as tables [111]) do not have special methods for appending them. They can be appended by general method AddItem [166].

1.1.2.2 Scrolling

Vertical scrolling

Documents in RichView can have almost unlimited height. But ranges of scrollbars are limited, and RichView cannot use pixels as scrolling units (small step of scrollbar). Scrolling units of RVScroller (ancestor of RichView) are referred here as "RVSU" (RichView Scrolling Units). By default, 1 RVSU = 10 pixels, but the component can increase it automatically during formatting if the range of vertical scrollbar exceeds 30,000. You can set a number of pixels in 1 RVSU with VSmallStep [159] property (but the component can increase it during formatting).

For example, you may want to set RVSU to the height of one line of the normal text. After assigning a new value to VSmallStep, you should reformat document using Format [182] method).

You can get or set position of vertical scrollbar using VScrollPos [512] property (in range 0..VScrollMax [512])

You can also use method ScrollTo [513] for scrolling to the specified vertical position, measured in pixels.

ScrollTo(y) is equivalent to VScrollPos := y div VSmallStep, so ScrollTo cannot scroll exactly to the specified position.

You can show and hide vertical scrollbar with VScrollVisible [512] property.

DocumentHeight [142] property is a vertical height of a document in pixels (height of scrollable area). For example, you can use this value to set height of RichView. DocumentHeight includes values of TopMargin [158] and BottomMargin [137].

You can get vertical coordinate in document for:

- *checkpoints* [30]
- hypertext link [35] (GetJumpPointY [197])
- any item (GetItemCoords [191])

Horizontal scrolling

Horizontal scrollbar appears automatically when necessary.

You can hide it with HScrollVisible [510] property.

You can get or set the position of horizontal scrollbar using HScrollPos [510] property (in range 0..HScrollMax [509])

You can use some properties to control width of a document (a width of scrollable area minus margins), thus controlling a horizontal scrolling (see LeftMargin [145], RightMargin [149])

By default, if the document does not contain wide images, controls or tables, text is wrapped to fit the client width of RichView minus margins.

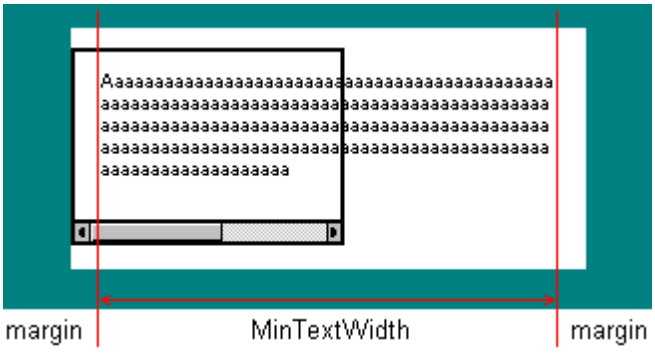
If there are pictures or controls, width of scrolling area is equal to

```
max(client width of RichView,
    "maximal width of pictures or controls" + "maximal left [451] + first line [451] +  right [454]
indent" + margins),
```

and text is wrapped to fit width of the scrolling area minus margins.

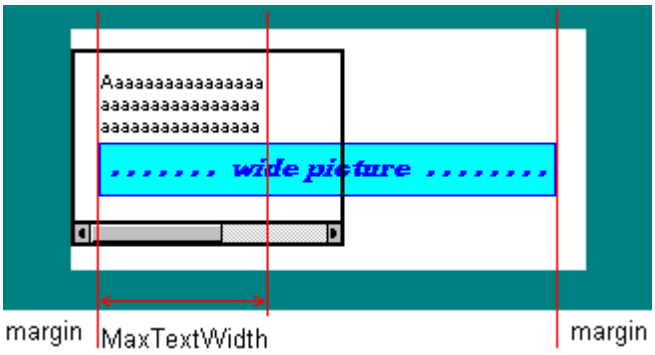
Text in paragraphs with no-wrap option [453] can also affect to width of scrolling area, in the same was as wide pictures or controls.

Use MinTextWidth [146] property to set the minimal width for text wrapping (text wraps when its length in pixels exceeds this value). Horizontal scrollable area cannot be less than MinTextWidth+margins (but can be greater because of wide pictures or wide client width of RichView). You can use MinTextWidth to speed up formatting when RichView is resized. Formatting is very quick if width of scrollable area is not changed. For example, if you set MinTextWidth to the size of screen, RichView will not perform full reformatting when user resizes it (well, it's not convenient to read text with horizontal scrollbar)



Explaining MinTextWidth

Use MaxTextWidth [146] property to set maximal width for text wrapping. This value does not affect width of scrollable area.



Explaining MaxTextWidth

A similar effect can be achieved using *rvoClientTextWidth* option from Options [146]. If this option is set, width for text wrapping is always equal to client width of RichView minus margins. It is useful if your document can have wide pictures or controls, or no-wrap paragraphs.

You can set no-wrap option [453] for paragraph style. If this option is set, this paragraph will not be wrapped automatically.

1.1.2.3 Selecting

User can select a part RichView document with the mouse (in RichViewEdit he/she also can use Shift +caret movement keys).

You can turn off/on a word selection mode, see RVStyle.SelectionMode [399].

Selection Appearance

Selected text has RichView.Style.SelTextColor/InactiveSelTextColor [393] color and RichView.Style.SelColor/InactiveSelColor [393] background.

By default, it is *clHighlightText* on *clHighlight* both for active (focused) and inactive windows.

See also RVStyle.SelectionStyle[399].

Operations

You can:

- disallow making selection (clear *rvoAllowSelection* in Options[146]); please do not use in editor!
- determine if the selection is not empty (SelectionExists[224]);
- deselect RichView (Deselect[181]; Invalidate);
- select the whole document (SelectAll[223]; Invalidate);
- get selection bounds (GetSelectionBounds[203]);
- select a part of the document (SetSelectionBounds[235];Invalidate);
- select the inserted control (SelectControl[223]);
- get the selection as a text (GetSelText, GetSelTextW[204]);
- get the selection as a Rich Text Format (SaveRTFToStream[220], SaveRTF[219]);
- get the selection as an image, if image is selected (GetSelectedImage[202]);
- save the selection to stream or file (SaveRVFToStream[220], SaveRVF[220]);
- copy the selection to the Clipboard in various formats (see RichView Clipboard Functions Overview[48])

In editor you can:

- delete selection (DeleteSelection[306]);
- select the current word[337] or line[337];
- cut selection to the Clipboard (CutDef[306]);

Note: All methods introduced in RichView which can change visual appearance require calling Invalidate or Refresh after them.

Note: All methods working with selection must be called only when the document is formatted.

See also...

Searching and replacing[49]

1.1.2.4 Clipboard

TCustomRichView and its descendants can copy information to the Clipboard as plain (ANSI and Unicode) text, image, RTF and RVF, and your own custom formats.

RichViewEdit can paste information from the Clipboard as text (ANSI and Unicode), image, RTF, and RVF (and your own formats).

Copying

The main method for copying to the Clipboard is CopyDef[177]. It can copy selection in one or more formats.

This method copies data in formats specified in RichView.Options[146]:

Option	Meaning
rvoAutoCopyText	if set, CopyDef copies selection as ANSI text
rvoAutoCopyImage	if set, CopyDef can copy selected image to clipboard
rvoAutoCopyUnicodeText	if set, CopyDef copies selection as Unicode text
rvoAutoCopyRTF	if set, CopyDef copies selection as RTF (Rich Text Format)
rvoAutoCopyRVF	if set, CopyDef copies selection as RVF

CopyDef is called automatically when the user presses Ctrl+Insert or Ctrl+C.

You can also use methods for copying:

- Copy 177 – copies selection to the Clipboard in all formats;
- CopyTextA 178 – copies the selection as ANSI text;
- CopyTextW 178 – copies selection as Unicode text
- CopyImage 177 – if there is a selected image, this method copies it to the Clipboard;
- CopyRVF 178 – copies selection in RVF format 59. This format has name 'RichView Format' and consist of two parts: <Size of RVF><RVF> where <Size of RVF> is 4-byte integer value (number of bytes in <RVF>), <RVF> is RVF data (see: About RVF 59, RVF Specification 77)
- CopyRTF 178 – copies selection as RTF (Rich Text Format).

SelectionExists 224 method answers to the question "does the selection contain some data (not empty)?"

OnCopy 240 event allows copying in your own formats.

Pasting

TRichView cannot paste. All these methods are methods of TRichViewEdit.

Pasting:

- PasteBitmap 329 – pastes bitmap from the Clipboard, if available;
- PasteMetafile 330 – pastes metafile picture from the Clipboard, if available;
- PasteGraphicFile 329 – pastes graphic file(s) the Clipboard, if available;
- PasteText 331 – pastes text from the Clipboard, if available;
- PasteRVF 331 – pastes RVF from the Clipboard, if available;
- PasteRTF 330 – pastes RTF from the Clipboard, if available;
- PasteURL 332 – pastes URL from the Clipboard, if available;
- Paste 328 – tries to paste as an RVF; if cannot – as an RTF, if cannot – as an URL, if cannot – as a text; if cannot – as graphic file(s), if cannot – as a bitmap; if cannot – as a metafile.AcceptPasteFormats 287 property is taken into account. This method is called automatically when user press Ctrl+V or Shift+Insert.

Testing is the Clipboard has the specific format:

- CanPaste 303 – "does the Clipboard contain any format what can be pasted in RichViewEdit?"
- CanPasteRVF 303 – "does the Clipboard contain RVF?"
- CanPasteRTF 303 – "does the Clipboard contain RTF?"

OnPaste 365 event allows to override the default pasting procedure. For example, you can paste data in different format, or you can allow pasting only some specific formats (for example, only a plain text).

Cutting

RichViewEdit.CutDef 306 copies the selection (like CopyDef) and then deletes it (like DeleteSelection 306). This method is called automatically when user presses Ctrl+X or Shift+Delete.

Windows messages

RichView processes *WM_COPY* message, RichViewEdit also processes *WM_PASTE, WM_CUT, EM_CANPASTE.*

See also...

Drag&drop 55

1.1.2.5 Searching and Replacing

The following main functions for text searching are available:

```
function TRichView128.SearchText222(s: String; SrchOptions: TRVSearchOptions642): Boolean;
function TRichViewEdit282.SearchText336(s: String; SrchOptions: TRVESearchOptions626): Boo
```

These methods search for the substring **s**. If found, they select it an make it visible (scroll the window to it).

Unicode note: ANSI and Unicode versions of these methods are available (SearchTextA and SearchTextW).

The methods of TRichView start searching from the current selection (if exists) or from the first (or the last, when searching up) visible item. If *rvsroFromStart* is included in SrchOptions, the methods start searching from the beginning (or the end, when searching up) of the document.

The methods of TRichViewEdit start searching from the caret position.

All the methods support the following options:

- searching up or down;
- case sensitive/insensitive searching;
- searching for the whole words;
- allowing matching in a single text item or in multiple items.

s may contain a tab character (#9). However, it makes sense only if the option for matching in multiple items is set.

The unit RVMisc.pas contains the functions to convert options of search dialogs:

```
function GetRVSearchOptions(fo: TFindOptions): TRVSearchOptions;
function GetRVESearchOptions(fo: TFindOptions):TRVESearchOptions;
```

It's easy to implement replacing of text in editor. SearchText selects the old substring, InsertText replaces the selected string with the new one.

1.1.2.6 Obtaining Items

There are two groups of methods that can be used to obtain information about items of RichView:

- **Get***Info** methods, introduced in RichView, can be used to get information about any item;
- **GetCurrent***Info** methods of RichViewEdit, return information about the "current" (at the caret position) item.

GetCurrent*Info** methods are equivalent to **Get***Info**(CurItemNo,...). But there is one important exception: when using tables, **GetCurrent***Info** methods return information about the item at the position of caret regardless its location (in root editor or in cell inplace editor).

Read-only property RichView.ItemCount returns the number of items.

```
function RichView.GetItemStyle(ItemNo: Integer): Integer;
```

This method returns style (type) of item with the specified index.

If the returned value>=0 then this is a text item and the returned value is an index in the collection of text styles (RichView.Style.TextStyles).

If the returned value <0 then this is a special item. Negative value could be equal to one of the following constants:

Constant	Value	Meaning
rvsBreak	-1	horizontal line
rvsPicture	-3	picture
rvsHotspot	-4	*hotspot*
rvsComponent	-5	Delphi control
rvsBullet	-6	*bullet*
rvsHotImage	-10	*hot-picture*
rvsListMarker	-11	list marker (bullet or numbering of paragraphs)
rvsTab	-12	tabulator
rvsTable	-60	table

See also: rvsXXX constants 663)

When you know a type of item you can get further information about it with the methods (**Get***Info** methods):

- GetBreakInfo 183,GetBulletInfo 184, GetHotspotInfo 189, GetPictureInfo 201, GetControlInfo 187, GetTextInfo 204.
- For item of any type your can get additional property: GetItemExtraIntProperty 193 and GetItemExtraStrProperty 193.
- For item of any type your can get "tag" (GetItemTag 195), "checkpoint" (GetItemCheckpoint 191)
- For any item you can get information about its paragraph: (GetItemPara 194, IsParaStart 208, IsFromNewLine 207).
- For any item you can get its text (text for text item, name for non-text item) (GetItemText 195)
- For any item you can get object representing it in a document (GetItem 190, usually used for tables 528)

In editor you can use the methods above or GetCurrentBreakInfo 308, GetCurrentBulletInfo 308, GetCurrentHotspotInfo 310, GetCurrentPictureInfo 315, GetCurrentControlInfo 309, GetCurrentTextInfo 316 (**GetCurrent***Info** methods), and GetCurrentTag 316, GetCurrentCheckpoint 309, GetCurrentItemText 313.

The arguments of these methods are similar to the arguments of **Add***Tag 43 methods for appending items 43 .

It should be noted that these methods return references to pictures and controls owned by RichView, not to their copy. Please use them wisely. For example, you can draw something on the returned picture but you should not destroy it. The preferred way to modify items of RichView is using **Set***Info 51 (or 51 **Set***InfoEd 51 for editor) 51. Exception: items with *rvepShared* 627 property.

A table in the position of the caret (or a top-level table containing the caret) can be obtained with the method GetCurrentItemEx 311.

1.1.2.7 Modifying Items

There are several groups of methods for modifying content of RichView:

- **Set***Info** methods, introduced in TRichView (viewer-style methods)
- **Set***InfoEd** methods of RichViewEdit (editor-style methods)
- **SetCurrent***Info** methods of RichViewEdit – the same as **Set***InfoEd** but work with the "current" (in the caret position) item.

SetCurrent*Info** methods are equivalent to **Set***InfoEd**(CurItemNo 288,...) But there is one important exception – when using tables 111, **SetCurrent***Info** methods modify item in the position of caret regardless its location (in the root editor or in cell inplace editor).

You cannot change a type of item (except for conversion pictures to hot-pictures 305 and vice versa 305)

First, you need to know a type of item. You can use RichView.GetItemStyle 194(ItemNo). Or you can use CurItemStyle 289 property of RichViewEdit for the item in the position of caret.

Next, you usually need to get information about the given item.

Finally, you can modify it.

Set***Info methods

The arguments of these methods are similar to the arguments of Add*** methods 43.

For the specified items you can use

- ·SetBreakInfo 226, SetBulletInfo 227, SetHotspotInfo 229 (call Invalidate to repaint RichView after them, reformatting is not necessary)
- ·SetPictureInfo 234, SetControlInfo 228 return Boolean value: "reformatting 182 is necessary?" (call Invalidate to repaint RichView after it, reformatting is not necessary if new picture or control has the same size).

You can also use:

- SetItemTag 231 method to change tags of items,
- SetCheckpointInfo 227 to change checkpoints of items,
- SetItemExtraIntProperty 230 to change values of additional integer properties,

- SetItemExtraStrProperty 231 to change values of additional integer properties,
- SetItemText 232 to change item text (visible text for text items, names for non-text items).

Please read the topic about "tags" 34 if you want to change tags of items.

The limitations of this version are:

- you can't change ImageList of *bullets* 96 and *hotspots* 97;
- you can't replace control with the different one;

It is not recommended to use these methods in editor, see Viewer vs Editor 71

Set***InfoEd methods (editor)

The same set of methods as **Set***Info** methods, with the same arguments, but they are editor-style methods.

SetBreakInfoEd 339, SetBulletInfoEd 339, SetHotspotInfoEd 349, SetPictureInfoEd 354, SetControlInfoEd 341;

also SetItemTagEd 352, SetCheckpointInfoEd 340, SetItemTextEd 352, SetItemExtraIntPropertyEd 350, SetItemExtraStrPropertyEd 351

SetCurrent***Info methods (editor)

The same as **Set***InfoEd** methods, but they work with the current (in position of caret) item.

SetCurrentBreakInfo 342, SetCurrentBulletInfo 342, SetCurrentHotspotInfo 345, SetCurrentPictureInfo 348, SetCurrentControlInfo 344,

also SetCurrentTag 349, SetCurrentCheckpointInfo 343, SetCurrentItemText 347, SetCurrentItemExtraIntProperty 345, SetCurrentItemExtraStrProperty 346.

Additional methods for controls

There are some special methods for resizing inserted controls in editor.

```
procedure ResizeControl 335(ItemNo, NewWidth, NewHeight: Integer);
procedure ResizeCurrentControl 335(NewWidth, NewHeight: Integer);
```

This is the preferred way to change size of the control. These methods reformat the affected part of document and repaint it.

If your control can sometimes change its size or move itself, you need to call

```
procedure AdjustControlPlacement 296(ItemNo: Integer);
procedure AdjustControlPlacement2 296(Control: TControl);
```

after such actions. These methods restore the correct position of the control and make all necessary reformatting and repainting if the control was resized.

1.1.3 Editing

1.1.3.1 RichViewEdit: Inserting Items in Position of Caret

These methods are useful for implementing "Insert" menu in your application. All these methods:

1. check ReadOnly 294 property (and do nothing in read-only mode);
2. delete the selected items;
3. insert item(s);
4. quickly reformat the changed part of the document;
5. repaint the editor;
6. generate OnChange 357 event, set Modified 293 to True.

The methods do nothing (and return True), if the selected text is modify/delete protected (or several cells of table are selected).

Inserting Text

These methods can insert several text items of the current text style (see CurTextStyleNo 290 property):

```
procedure InsertText 327(const text: String; CaretBefore: Boolean=False);
```

ANSI (InsertTextA) and Unicode (InsertTextW) versions of InsertText are available.

```
function InsertTextFromFile 328(const FileName: String):Boolean;
function InsertTextFromFileW 328(const FileName: String):Boolean;
function InsertOEMTextFromFile 321(const FileName: String):Boolean;
```

These methods insert one text item (string must not contain line break, page break and tab characters) with the specified tag:

```
function InsertStringTag 326(const s: String; Tag: TRVTag 645): Boolean;
```

ANSI (InsertStringATag) and Unicode (InsertStringWTag) versions of InsertStringTag are available.

Inserting Non-Text Items

These methods insert one item of the specified type (parameters are similar to parameters of **Add*** 43** methods 43):

```
function InsertControl 318(const Name: TRVAnsiString 623;
  ctrl: TControl; VAlign: TRVVAlign): Boolean;
function InsertPicture 322(const Name: TRVAnsiString 623;
  gr: TGraphic; VAlign: TRVVAlign): Boolean;
function InsertHotPicture 319(const Name: TRVAnsiString 623;
  gr: TGraphic; VAlign: TRVVAlign): Boolean;
function InsertBreak 317(Width: Byte;
  Style: TRVBreakStyle; Color: TColor): Boolean;
function InsertBullet 317(ImageIndex: Integer;
  ImageList: TCustomImageList): Boolean;
function InsertHotspot 319(ImageIndex, HotImageIndex: Integer;
  ImageList: TCustomImageList): Boolean;
```

Inserting Files and Streams

These methods insert RVF 59 file or stream

```
function InsertRVFFromStreamEd 325(Stream: TStream):Boolean;
function InsertRVFFromFileEd 324(const FileName: String):Boolean;
```

These methods insert RTF (Rich Text Format) file or stream

```
function InsertRTFFromStreamEd 323(Stream: TStream):Boolean;
function InsertRTFFromFileEd 323(const FileName: String):Boolean;
```

These methods insert text of the current text style (see CurTextStyleNo 290 property):

```
function InsertTextFromFile 328(const FileName: String):Boolean;
function InsertTextFromFileW 328(const FileName: String):Boolean;
function InsertOEMTextFromFile 321(const FileName: String):Boolean;
```

Inserting Tables and Other Items

The most general method for inserting items is

```
function InsertItem 320(const Name: String; Item: TCustomRVItemInfo): Boolean;
```

It's recommended to use this method only for tables 98, labels 101, numbered sequences 102, endnotes 103, footnotes 104, references to parent footnotes and endnotes 108, and custom item types 111.

See also...

Methods for pasting from the Clipboard 48.

1.1.3.2 Undo and Redo

(Introduced in version 1.3)

All methods for document editing that introduced in editor (TRichViewEdit[282]) can be undone/redone (see editor-style methods).

All methods for document modification that introduced in TRichView[128] can't be undone/redone (see viewer-style methods). If you wish to modify document with these methods, you must clear undo buffer before.

All methods and properties described in this topic are methods and properties of TRichViewEdit.

```
property UndoLimit[295]: Integer
```

Property UndoLimit sets the capacity of undo buffer. It can be used to disable undo/redo feature.

Basic undo/redo

Undo/redo is available in editor even without any effort from your side.

User can undo the last operation by pressing Ctrl + Z or Alt + Backspace.

User can redo the last undone operation by pressing Shift + Ctrl + Z or Shift + Alt + Backspace.

Undo/redo buffers will be cleared automatically when you call Clear[175] method, or when you use methods for loading documents (which call Clear inside).

You can clear undo and redo buffer yourself using method ClearUndo[305].

Implementing menu/toolbar buttons for undo/redo

You can undo/redo operations from the application code, using methods Undo[355] and Redo[332].

You can change enabled/disabled state of menu items or buttons comparing the returned values of functions UndoAction[355] and RedoAction[333] with *rvutNone* (*rvutNone* – menu for undo/redo should be disabled; other values – enabled).

Call these methods inside OnChange[357] event.

You can modify caption/hint of menu item or button to display the name of action that will be undone/redone next. A value identifying this undo/redo action can be obtained using UndoAction[355]/RedoAction[333].

There are no predefined names for standard undo/redo actions. The Editor demo (**Demos\Delphi\Editors\Editor 1**) has the suggested names of actions in RVUndoStr.pas.

rvutCustom constant is a special case, see below. For this value, use UndoName[355]/RedoName[333] methods.

Grouping operations for undo

You can set the mode so that all subsequent operations will be considered by the editor as a whole, and they will be undone (and redone) in one step.

First, you need to give name to this undo action. Name can be standard (not a name really, but a type of undo action) or custom.

Use BeginUndoGroup[302] to give a standard name, and BeginUndoCustomGroup[302] to give a custom name.

Next, you need to call SetUndoGroupMode[355].

```
procedure TRichViewEdit.BeginUndoGroup(UndoType: TRVUndoType);
```

– begins a new undo action. It must be followed by SetUndoGroupMode(True) ... SetUndoGroupMode(False).

UndoType can be any value of type TRVUndoType[647], except for rvutNone and rvutCustom.
This value will be returned by UndoAction/RedoAction when undo/redo.

```
procedure TRichViewEdit.BeginUndoCustomGroup(const Name: String);
```

– the same as BeginUndoGroup, but allows to set your own name for the undo action. The undo action will have type = *rvutCustom*.

Name of it will be returned by UndoName/RedoName.

```
procedure TRichViewEdit.SetUndoGroupMode(GroupUndo: Boolean);
```

– sets mode where all subsequent actions will be undone and redone as one. It *must* be preceded by either BeginUndoGroup or BeginUndoCustomGroup. Typing can not be grouped. Operations in different cell inplace editors cannot be grouped.

These methods must be called for the editor where these operations are performed (usually TopLevelEditor[295])

Example (inserting picture and resizing it 100x100):

```
MyRichViewEdit.TopLevelEditor.BeginUndoGroup(rvutInsert);
MyRichViewEdit.TopLevelEditor.SetUndoGroupMode(True);
try
  if MyRichViewEdit.InsertPicture[322](...) then begin
    MyRichViewEdit.SetCurrentItemExtraIntProperty[345](
      rvepImageHeight, 100, True);
    MyRichViewEdit.SetCurrentItemExtraIntProperty(
      rvepImageWidth, 100, True);
finally
  MyRichViewEdit.TopLevelEditor.SetUndoGroupMode(False);
end;
```

Each call of SetUndoGroupMode(True) should be followed by exactly one call of SetUndoGroupMode(False).

Calls can be nested.

See also...

Methods introduced in TRichView and undo: see Viewer vs Editor[71]

1.1.3.3 Drag and Drop

You can drag data from TRichView[128] or TRichViewEdit[282]. You can drop data to TRichViewEdit[282].

Drag&drop is possible not only inside the same editor, but between different controls and even between different applications.

For example, you can drop files from Windows Explorer, hyperlinks, text and images from Internet Explorer or Word.

Drag&drop feature requires Delphi 3 or newer, or C++Builder 3 or newer (it is disabled for Delphi 2 and C++Builder 1).

Dragging from TRichView (or TRichViewEdit)

TRichView provides data in the following formats:

- RichView Format[59] ('RichView Format')
- RTF ('Rich Text Format')
- ANSI text (CF_TEXT)
- Unicode[64] text (CF_UNICODETEXT)
- bitmap (CF_BITMAP), if TBitmap picture[90] is selected
- metafile (CF_ENHMETAFILE), if TMetafile picture[90] is selected.

You can disallow dragging from the given TRichView by including *rvoDisallowDrag* in the Options[146] property.

TRichView cannot initialize dragging of inserted controls[94]. If you want to drag them, use BeginOleDrag[174] method.

Dragging to TRichViewEdit

AcceptDragDropFormat[287] lists formats that can be accepted.

By default all formats are accepted except for URL (because it requires processing in OnReadHyperlink[248])

Dropped object can contain data in multiple formats. The priority is: RVF, then RTF, then URL, then text, then bitmap, then metafile, then files.

If the current text style[290] is Unicode[446], Unicode text format has higher priority than ANSI text format, and vice versa.

The following events may occur:

- OnReadHyperlink[248];
- OnDropFiles[359].

You can customize drag&drop process using the following low level events:

- OnOleDragEnter[361];
- OnOleDragOver[362];
- OnOleDragLeave[362];
- OnOleDrop[363].

See also...

Clipboard[48]

1.1.4 Graphics

1.1.4.1 Animated Images

Overview

TRichView can animate inserted images[90].

AnimationMode[135] property defines when animation starts. By default, AnimationMode[135]=*rvaniManualStart*.

rvaniManualStart is used by default because *rvaniOnFormat* may cause problems in some existing applications (created before implementing this feature), because animation must be stopped when you call viewer-style methods like DeleteItems[179]. If you simply load and edit documents, set this mode to *rvaniOnFormat*.

Methods:

- StartAnimation[237] starts all animations;
- StopAnimation[237] stops all animations;
- ResetAnimation[215] rewinds all animations to the first frame.

Supported Animations

1. TGifImage included in Delphi 2007, 2009 or newer

Include RVGifAnimate2007 unit in your project to enable this animation.

2. TGifImage by Anders Melander

Downloads:

- ***http://www.torry.net/vcl/graphics/gif/gifimage.exe*** (original);
- ***http://www.trichview.com/resources/thirdparty/gifimage.zip*** (update, necessary).

Include RVGifAnimate unit in your project to enable this animation.

3. RVJvGifAnimate for TJvGifImage from JEDI's JVCL

Dowloads: ***http://jvcl.sourceforge.net***.

4. TBitmap

Bitmap is sliced into animation frames (*rvepImageWidth* x *rvepImageHeight*) arranged in rows and columns. Animation is enabled if you set nonzero *rvepAnimationInterval* (animation delay in 1/100 of second), see TRVExtraItemProperty[627] type. If one of *rvepImageWidth* and *rvepImageHeight* is not defined (zero), frame

width/height is assumed to be equal to the image width/height. If both of them are not defined, animation is not played.

1.1.4.2 Smooth Image Scaling

Scaling

TRichView can display scaled images using an advanced method producing smooth scaled images.

The following images are affected:

- TCustomRichView.BackgroundBitmap[136] (if BackgroundStyle[136]=*bsStretched*);
- pictures[90] and *hot-pictures*[93] (when stretched);
- table.BackgroundImage[530] (if BackgroundStyle[531]=*rvbsStretched*);
- table cells' BackgroundImage[567] (if BackgroundStyle[568]=*rvbsStretched*).

By default, only non-transparent bitmaps and JPEG images are scaled, and not greater to 3000 pixels in width or height. These settings can be changed using RVThumbnailMaker[664] object.

Smooth scaling is applied when displaying pictures on the screen. It is not used when printing or displaying in TRVReportHelper[503].

ScaleRichView note: in TSRichViewEdit, smooth scaling is applied to the same images, but they do not need to be stretched: they can be scaled when the whole page is scaled.

Caching

Smooth scaling is not a very fast operation, so resulting images are stored for reusing in future. Each TRichView component can store up to RichViewMaxThumbnailCount[655] scaled images (+ a scaled version TCustomRichView.BackgroundBitmap[136], which is always stored).

1.1.4.3 Semitransparent Objects

Documents may contain objects having transparent and semitransparent areas, such as:

- pictures[90] (for example, Gif, Png, icons, Bitmaps with alpha-channel)
- paragraph background (with specified Opacity[467])
- table[98] background (that may include transparent pictures[530] and semitransparent[540] color filling)
- table cell background (that may include transparent pictures[567] and semitransparent[571] color filling)
- sidenote[105] and text box[107] background (with specified Opacity), displayed only while printing in TRVPrint[479] and in ScaleRichView.

Drawing on the screen

When drawing on the screen, the component uses the appropriate methods for semitransparent drawing. Some special code is used to draw backgrounds of animations[56] and backgrounds of inplace editors of table cells.

Known problems: TPngObject does not support semitransparency when displayed rotated[573].

Printing in TRVPrint[479]

The most of printers do not support semitransparent drawing, so TRVPrint draws semitransparent objects in the following steps:

1. it creates a bitmap,
2. it draws a background below the transparent object in this bitmap,
3. it draws this object in this bitmap,

4. it prints this bitmap.

TRVPrint has a limited support of transparency in sidenotes and text boxes (when drawing on the step 2):

- you cannot see a sidenote or a text box through a transparent area
- through a transparent area in a sidenote or a text box, you can see only a page background (but not any content)

TRVPrint does not support textured backgrounds in rotated objects (a plain color is used instead).

Drawing and printing in TRVReportHelper[503]

When printing, TRVReportHelper uses the same method (using bitmaps to draw semitransparent objects) as TRVPrint.

When drawing on the screen, TRVReportHelper can draw transparently or can use bitmaps, depending on AllowTransparentDrawing[504] property.

Drawing and printing in ScaleRichView

In TSRichViewEdit, drawing on the screen is similar to drawing in TRichViewEdit: the component uses the appropriate methods for semitransparent drawing.

When printing, like TRVPrint, TSRichViewEdit uses a bitmap to draw semitransparent areas. However, the step 2 (drawing a background below the transparent object in a bitmap) is implemented differently. When drawing a background below the transparent object, TSRichViewEdit draws all page layers below the object. Because of this, you can see text boxes and sidenotes through the transparent areas, and you can see main document through transparent areas in text boxes and sidenotes.

TSRichViewEdit does not have a limitation on printing backgrounds of rotated objects.

Bitmaps created to draw semitransparent areas in ScaleRichView are created in resolution 96 dpi. Because of this, when printing on high resolution printers, semitransparent areas are printed more roughly than surrounding text.

1.1.5 Saving and Loading

1.1.5.1 Saving and Loading

RichView has the following methods for export:

- SaveHTML[216], SaveHTMLEx[217]: saving HTML files (also SaveHTMLToStream[218], SaveHTMLToStreamEx[218])
- SaveText[221]: saving plain ANSI text in file; SaveTextW[221]: saving plain Unicode text; SaveTextToStream, SaveTextToStreamW[221] – the same for streams (can save the whole document or only selection);
- SaveRTF[219],SaveRTFToStream[220] – exporting the whole document or the selected part to file or stream as RTF (Rich Text Format);
- SaveDocX[215],SaveDocXToStream[216] – exporting the whole document or the selected part to file or stream as DocX (Microsoft Word format);
- SaveRVF[220],SaveRVFToStream[220] – saving the whole document or the selected part to RVF (RichView Format[59]) file or stream;

RichView has the following methods for import:

- LoadText[212]: reading plain ANSI text; LoadTextW[212]: reading plain Unicode text;
- LoadTextFromStream, LoadTexFromStreamtW[213] - the same for streams instead of files;
- LoadRTF[209], LoadRTFFromStream[210]: reading RTF (Rich Text Format) file or stream.
- LoadRVF[211], LoadRVFFromStream[211]: reading RVF (RichView Format[59]) file or stream;

See also: InsertRVFFromStream[206], AppendRVFFromStream[173].

Note: All methods for reading contents, introduced in RichView, are viewer-style methods.

RichViewEdit has the following additional methods for import (insertion):

- InsertRVFFromFileEd 324, InsertRVFFromStreamEd 325 – like InsertRVFFromFile/InsertRVFFromStream, but editor-style methods.
- InsertRTFFromFileEd 323, InsertRTFFromStreamEd 323 – inserting RTF (Rich Text Format) file or stream.
- InsertTextFromFile 328 – editor-style method;
  InsertOEMTextFromFile 321 – the same method for files in DOS code page;
  InsertTextFromFileW 328 – the same method for Unicode files.

There are also method for pasting data from the Clipboard 48.

Events on export

- HTML: OnHTMLSaveImage 241, OnSaveImage2 260, OnSaveComponentToFile 256, OnSaveHTMLExtra 259, OnSaveItemToFile 261, OnSaveParaToHTML 263, OnWriteHyperlink 267.
- RTF: OnSaveComponentToFile 256, OnSaveRTFExtra 263, OnSaveItemToFile 261, OnWriteHyperlink 267.
- DocX: OnSaveComponentToFile 256, OnSaveDocXExtra 257, OnSaveItemToFile 261, OnWriteHyperlink 267.
- Text: OnSaveComponentToFile 256, OnSaveItemToFile 261

Events on import

- RTF: OnImportPicture 242, OnReadHyperlink 248, OnStyleTemplatesChange 266.

Common events

- OnProgress 248

See also...

Saving HTML 62
Reading/writing RVF 59
Export of tables 123
Unicode in RichView 64

1.1.5.2 RVF (RichView Format)

What is RVF?

RVF (RichView Format) is a file format specially designed for saving/loading TRichView documents. It is quite simple and compact format. It can be text file or binary file (defined by RichView.RVFOptions 152).

This format is used:

- in TDBRichView 373, TDBRichViewEdit 377 to save their contents to binary fields;
- RVF can be copied to the Clipboard; editor can paste RVF;
- data can be dragged&dropped 55 as RVF;
- you can load/save RVF with the methods: LoadRVFFromStream 211, InsertRVFFromStream 206, AppendRVFFromStream 173, LoadRVF 211, SaveRVFToStream 220, SaveRVF 220.

All RichVew document features can be saved in RVF (including text, images, bullets, hotspots, controls, checkpoints, tables). Collections of text, paragraph, list styles,style templates, margins and background can be saved optionally (see below).

The main things you need to know about RVF

- main settings affecting RVF saving and loading can be changed in the TRichView component editor 134;
- in order to load *bullets* 96, *hotspots* 97 and list levels 472 with imagelist from RVF you need to process OnRVFImageListNeeded 252 event;
- you can save pictures 90 and controls 94 in binary mode or text mode (binary mode is much more efficient and compact);
- you can save pictures and controls in RVF, or create them on request when loading;
- if you load pictures and controls in RVF you need to register them with RegisterClasses procedure (you do not

need to register TBitmap, TMetafile, TIcon and TJpegImage)
- formatting of text[404] and paragraph[398] (i.e. styles) is saved in RVF documents optionally, if *rvfoSaveTextStyles* and *rvfoSaveParaStyles* are included in RVFOptions[152];
- style templates are saved in RVF, if *rvfoSaveTextStyles* and *rvfoSaveParaStyles* are included in RVFOptions[152], and UseStyleTemplates[158]=*True*.

RVF and Image Lists

RichView can save and load all document content itself, but there is an exception: *bullets*[96], *hotspots*[97] and list markers having imagelist picture in list levels[472].

When loading, RichView needs to know which ImageList to use for the given *bullet* or *hotspot*. It asks the application about it by calling OnRVFImageListNeeded[252] event. You need to process this event if you want to load *bullets* and *hotspots*.

When saving, RichView stores Tag properties of ImageList with *bullet*[96] or *hotspot*[97]. When reading, RichView calls OnRVFImageListNeeded event and pass this value to it.

Example

If you have two ImageLists (ImageList1 and ImageList2) that you use for *bullets* in MyRichView, you can:
- set ImageList1.Tag to 1, set ImageList2.Tag to 2,
- process MyRichView.OnImageListNeeded event::

```
procedure TMyForm.MyRichViewRVFImageListNeeded(
  Sender: TCustomRichView; ImageListTag: Integer;
  var il: TCustomImageList);
begin
  case ImageListTag of
    1:
      il := ImageList1;
    2:
      il := ImageList2;
  end;
end;
```

DO NOT CONFUSE IMAGELIST.TAG WITH RICHVIEW ITEMS' TAGS[34]. IN THIS CASE RICHVIEW USES TIMAGELIST.TAG PROPERTY.

RichView does not own any ImageList, it never destroys them. It just holds references to them.

If you want to load pictures and/or controls from RVF you need to register them with RegisterClasses procedure (you need not to register TBitmap, TMetafile, TIcon and TJpegImage - they are already registered by RichView)

RVF and Pictures & Controls

You can save full information about pictures and inserted controls in RVF, but you can also use advanced feature: you can save pictures/controls yourself and include in RVF only some "identifiers" that can be used to create pictures/controls when needed (on loading). Two values can be uses as these "identifiers": RichView items tags and names.

These options do not affect loading from RVF. RichView autodetects these modes (binary/text, indices/names of styles).

When loading, you can use OnRVFPictureNeeded[252] and OnRVFPictureNeeded[252] events.

Example

```
procedure TMyForm.MyRichViewRVFControlNeeded(
  Sender: TCustomRichView; Name: String; const Tag: TRVTag[645];
  var ctrl: TControl);
begin
  case StrToInt(Tag) of
```

```
    1:
      begin
        ctrl := TButton.Create(nil);
        TButton(ctrl).Caption := Name;
      end;
    2:
      begin
        ctrl := TEdit.Create(nil);
        TEdit(ctrl).Text := Name;
      end;
  end;
end;
```

As you can see, RichView passes two parameters that can help you to identify your control:

- Tag: TRVTag [645] – RichView item tag (do not confuse with tag properties of TComponent descendants);
- Name: String – string that can be associated with any control or picture in RichView.

Example

```
// this example assumes that images may be only bitmaps
procedure MyRichViewRVFPictureNeeded(Sender: TCustomRichView[128];
    const ItemName: String; Item: TRVNonTextItemInfo[526];
    Index1, Index2: Integer; var gr: TGraphic);
begin
  gr := TBitmap.Create;
  if (Index1>=0) and (Index2>=0) then
    gr.LoadFromFile((Item as TRVTableItemInfo[528]).Cells[537][Index1, Index2].Backg
  else if Item is TRVTableItemInfo then
    gr.LoadFromFile(TRVTableItemInfo(Item).BackgroundImageFileName[531])
  else if Item is TRVGraphicItemInfo then
    gr.LoadFromFile(TRVGraphicItemInfo[526](Item).ImageFileName[630]);
end;
```

Options for Saving and Loading. Warnings

RichView.RVFOptions [152] define options for RVF saving and loading.

After loading you can check RichView.RVFWarnings [155] property. Depending on RichView.RVFOptions some flags in RichView.RVFWarnings can be errors or warning:

Units of measurement

RVF file may be written with different units of measurement (pixels or twips [74]) than specified in Style [385].Units [404] of TRichView which loads this file.

When inserting RVF, lengths from RVF are converted to Style [385].Units [404].

When loading RVF, the control works according to *rvfoCanChangeUnits* option. If it is excluded from RVFOptions [152], lengths from RVF are converted to Style [385].Units [404]. If it is included, and RVF contains either text or paragraph styles, existing styles are converted to units read from RVF (using Style [385].ConvertToDifferentUnits [405] method), so lengths from RVF are read without conversion.

If a conversion of units is performed while reading RVF, *rvfwConvUnits* is included in RVFWarnings [155].

See also...

RVF Specification [77]

RichView properties:

- Options [146],
- RVFOptions [152],

- RVFWarnings 155;
- UseStyleTemplates 158;
- StyleTemplateInsertMode 156.

RichView events:

- OnRVFImageListNeeded 252,
- OnVFControlNeeded 251,
- OnRVFPictureNeeded 252,
- OnControlAction 239 (ControlAction=*rvcaAfterRVFLoad*),
- OnItemAction 244 (ItemAction=*rviaInserting* and *rviaInserted*);
- OnStyleTemplatesChange 266.

RichView methods:

- LoadRVFFromStream 211, InsertRVFFromStream 206, AppendRVFFromStream 173, LoadRVF 211,
- SaveRVFToStream 220, SaveRVF 220,
- Copy 177, CopyDef 177, CopyRVF 178.

RichViewEdit methods:

- InsertRVFFromStreamEd 325, InsertRVFFromFileEd 324,
- Paste 328, PasteRVF 331, CanPasteRVF 303, CutDef 306.

1.1.5.3 Export to HTML

Overview

RichView has methods for saving HTML files. There are no methods for reading HTML files in current version (use free addon: ***TrvHtmlImporter***).

RichView saves HTML file and a set of image files.

By default, images are saved as Jpegs (*.jpg).

If you do not want to use TJpegImage class, you can define RVDONOTUSEJPEGIMAGE conditional define when compiling, and images will be saved as bitmaps.

You can specify graphic formats that must not be converted to Jpegs using RVGraphicHandler 662 .RegisterHTMLGraphicFormat.

Besides, you can save images yourself in custom formats using OnHTMLSaveImage 241 or OnSaveImage2 260 event (see demo in **Demos\Delphi\Assorted\Save HTML**)

There are two ways to save HTML in RichView:

- saving using <font>, <b>, <i> and similar html tags,
- saving using CSS (Cascading Style Sheets).

HTML files which were saved with the first way can be read by larger number of programs (including TrvHtmlImporter).

HTML files with CSS retain much more formatting properties of RichView document (they look almost exactly like RichView content when using a capable browser) and have much more elegant and compact HTML code. Such files can be successfully viewed for example with MS Internet Explorer 4+, Netscape Navigator 4+, MS Word 2000+; but when viewed, for example, with MS Word 97 they lose all formatting. TRVHtmlImporter does not support CSS generally, but it can read HTML with CSS files saved by TRichView.

Saving Using <font> Tags

```
function RichView.SaveHTML(const FileName,Title,
  ImagesPrefix: String; Options: TRVSaveOptions):Boolean;
```

FileName – file name.

Title – title of document (tag <title>).

Return value – "saving was successful?"

ImagesPrefix – first part of images file names. For example, if **ImagesPrefix**='img' then images will be saved as **img1.jpg**, **img2.jpg** and so on (or **img1.bmp**, **img2.bmp** and so on, see above).

If *rvsoOverrideImages* in **Options** then all files with the same names will be overridden. Otherwise, the component will make unique file names for images.

Saved jpegs or bitmaps are not transparent. When converting to jpeg or bitmap, transparent color of icons or metafiles converted to the background color of RichView.

You can change saving of images using OnHTMLSaveImage 241 or OnSaveImage2 260 events.

Checkpoints 30 are saved as where # is an index of *checkpoint*. If *rvsoUseCheckpointsNames* is in **Options**, *checkpoint* name is used instead.

RichView uses Charset property of 0th text style to save language of HTML file (for Delphi 3+). There is also an option to save Unicode (UTF-8) HTML files.

When saving with **SaveHTML** method, many formatting properties are lost (such as text and paragraph background colors, borders, hover color, all margins/indents, font sizes are converted to 7 available HTML font sizes). Centered and stretched background bitmaps become tiled.

Saving Using CSS

```
function RichView.SaveHTMLEx(const FileName, Title, ImagesPrefix,
  ExtraStyles, ExternalCSS, CPPrefix: String;
  Options: TRVSaveOptions):Boolean;
```

This method is not an extension of SaveHTML method – this is a completely different method. It saves HTML using style sheets (CSS). CSS can be saved inside HTML file or this method can place only a reference to external CSS file into this HTML file

External CSS file can be saved using SaveCSS 408 method of RichView.Style:

You can save your style sheet one time and use it for all saved HTML files (if they all use the same collection of styles). This method allows you to reduce size when saving several HTML files with the same text and paragraph styles.

Font families are stored in CSS for the most frequently used fonts. For example, if font name is 'Arial', the string *'Arial', 'Helvetica', sans-serif* will be saved.

ExtraStyles is a string that can contain additional styles for your own use. RichView will not use these styles, but you can use them when saving, for example, controls and URLs.

CPPrefix is a first part of name used when saving *checkpoint* 30. If **CPPrefix**='' then RichView uses standard name ('RichViewCheckPoint'). For example, if **CPPrefix**='label' then checkpoints will be saved as where # is an index of *checkpoint*. If *rvsoUseCheckpointsNames* is in **Options**, *checkpoint* name is used instead.

Usually you do not need to use advanced parameters and can write:

```
MyRichView.SaveHTMLEx(MyFileName, MyTitle, MyImagesPrefix,
  '', '', '', []);
```

Saved HTML looks exactly like RichView with the exceptions:

- saving centered background pictures instead of stretched;
- if VShift 443 property of text style >0 then this style saved as superscript, <0 as subscript;
- all pictures becomes jpegs or bitmaps, so transparency is lost (transparent pixels are replaced with color of picture's paragraph or RichView itself), if you do not use OnHTMLSaveImage 241 or OnSaveImage2 260 events or if you did not define their formats as html graphic format 672.
- only "normal" (not "hot") *hotspots* 97 pictures are saved;
- border styles converted to available set for HTML;
- negative BorderOffsets can be saved, but browsers understand them as zero offsets;
- paragraph background offsets and border offsets can't be saved separately; maximal values of these offsets are saved;
- old browsers (such as Netscape Communicator 4) do not support subscripts/superscripts, partial borders.

SaveHTMLEx[217] saves all text of "Symbol" font as a set of HTML character entities or Unicode codes. For example, the text 'a+b=g' of "Symbol" font (α+β=γ) is saved as 'α+β=γ'

Pros:

- correct view not only in IE, but also in Opera and Mozilla;
- correct view on non-Windows platforms

Cons:

- characters may look slightly different;
- some exotic characters are lost.

Common for the Both Methods

By default, items tags[34] are saved as targets of hypertext links. You can customize saving using the event OnWriteHyperlink[267].

By default, inserted controls are not saved. However, you can save them in the event OnSaveComponentToFile[256].

You can save additional information in HTML in the events: OnSaveHTMLExtra[259], OnSaveParaToHTML[263].

You can insert HTML codes[438] in text.

You can completely change saving of items of any type in HTML using OnSaveItemToFile[261] event.

There is a set of methods for saving HTML files to stream: TRichView.SaveHTMLToStream[218], TRichView.SaveHTMLToStreamEx[218], TVStyle.SaveCSSToStream[409].

There is also a possibility to save only part of HTML document (opening part, middle part, closing part) – useful if you use HTML output of RichView only as a part of HTML document.

Known problems:

Numeration of checkpoints starts from 0 in each table cell. So, it's recommended to use checkpoint names as identifiers (add *rvsoUseCheckpointsNames* in **Options**).

Some ideas what may be implemented in future

- to associate HTML tags with text styles (such as <H1>, <B>) and use them when saving HTML file; this method gives very simple compact HTML and almost all browsers will understand it; Though it can look not similar to original RichView document.

See also...

- RichView methods for saving and loading[58];
- Export of tables[123];
- Typed constants related to HTML export[652];
- Example how to save MIME-encoded files (*.eml or *.mht), how to send HTML e-mail using TNMSMTP or TIdSMTP components: ***http://www.trichview.com/forums/viewtopic.php?t=11***.

1.1.6 Internationalization

1.1.6.1 Unicode in TRichView

Introduction

Unicode is a worldwide character-encoding standard. Unicode simplifies localization of software and improves multilingual text processing. By implementing it in an application, a developer can enable the application with universal data exchange capabilities for global marketing, using a single binary file for every possible character code. Because each Unicode character is 16 bits wide (in UTF-16 encoding), it is possible to have separate values for up to 65,536 characters. Unicode-enabled functions are often referred to as "wide-character" functions.

For Delphi 2009 or newer, Unicode is a default encoding for strings.

Unicode strings are referred here as **'Unicode'**

Single-byte strings are referred here as **'ANSI'** (for simplicity)

Unicode and ANSI Text in TRichView

Not all strings in TRichView are Unicode strings.

Text in text items [88] can be Unicode or ANSI depending on Unicode [446] property of text style.

Text (item name) of non-text items is always ANSI.

The following text depends on version of Delphi/C++Builder (Unicode for Delphi/C++Builder 2009 or newer, ANSI for older versions):

- *tag* [34] strings [34];
- extra item properties [630];
- names of *checkpoints* [30];
- visible text in labels [101], numbered sequences [102], endnotes [103], footnotes [104];
- live spelling [69] interface;
- text in list markers [99];
- hypertext targets (OnReadHyperlink [248], OnWriteHyperlink [248]);
- and others.

Main Limitations of the Current Implementation

You must prevent conversion of Unicode to double-byte character set (DBCS) strings, used for representation of characters in Asian languages, because DBCS is not supported by RichView. The only exception (where conversion is ok) is exporting and saving (because in these cases DBCS text will not be used in TRichView).

How to Enable Unicode. Using Both ANSI and Unicode

Set Unicode [446] property of text style to *True*. Important: document must be empty when changing this property. TRichViewEdit initially has one empty string, so it is not completely empty, call Clear [175] before changing this property. The default value of this property is *True* for Delphi/C++Builder 2009 or newer.

Document can contain both Unicode and ANSI text (in different styles).

So, you can mix ANSI and Unicode text. Of course, you can use only ANSI or only Unicode styles. This is even recommended.

How to Make Unicode Editor (Without ANSI Text)

1. Set Unicode [446] property to True for all TextStyles in TRVStyle. Important: document must be empty when changing this property. TRichViewEdit initially has one empty string, so it is not completely empty, call Clear [175] before changing this property. The default value of this property is True for Delphi/C++Builder 2009 or newer.
2. Set RichViewEdit1.RTFReadProperties [151].UnicodeMode [274] = *rvruOnlyUnicode* (this is the default value of this property for Delphi/C++Builder 2009 or newer).
3. Many methods working with text have 3 versions:

- with TRVUnicodeString [647] parameters (finished with -W, for example SearchTextW);
- with TRVAnsiString [623] parameters (finished with -A, for example SearchTextA);
- with String parameters (for example, SearchText).

For Delphi/C++Builder versions prior to 2009, use TRVUnicodeString [647]-methods. For Delphi/C++Builder, you can use either TRVUnicodeString [647]-methods or String-methods. Avoid using TRVAnsiString [623]-methods to prevent conversion between Unicode and ANSI text.

These methods include the following methods of TRichView (methods names without -A and -W are listed):

- AddNLTag [161] and its versions;
- AddTextNL [171];

- GetItemText 195;
- GetSelText 204;
- GetWordAt 205;
- SearchText 222;
- SetItemText 232.

These methods include the following methods of TRichViewEdit (methods names without -A and -W are listed):

- GetCurrentItemText 313;
- InsertStringTag 326;
- InsertText 327;
- PasteText 331;
- SearchText 336;
- SetItemTextEd 352;
- SetCurrentItemText 347.

4. Existing non-Unicode RVF 59 documents must be converted to Unicode by calling ConvertToUnicode after loading them (see below).

This step is not necessary for Delphi/C++Builder 2009: all text styles in RVF documents saved by applications compiled with older version of Delphi/C++Builder are converted to Unicode automatically.

It's safe to call this procedure for Unicode documents – it will do nothing.

Example: converting TRichView document to Unicode

```
uses CRVData, RVItem, RVUni;
// this code uses some undocumented methods
procedure ConvertRVToUnicode(RVData: TCustomRVData 608);
var i,r,c, StyleNo: Integer;
    table: TRVTableItemInfo 528;
begin
  for i := 0 to RVData.ItemCount 144-1 do begin
    StyleNo := RVData.GetItemStyle 194(i);
    if StyleNo>=0 then begin
      if not RVData.GetRVStyle.TextStyles 404[StyleNo].Unicode 446 then begin
        RVData.SetItemTextR(i, RVU_GetRawUnicode(RVData.GetItemTextW 195(i)));
        Include(RVData.GetItem(i).ItemOptions, rvioUnicode);
      end;
      end
    else if RVData.GetItemStyle(i)=rvsTable then begin
      table := TRVTableItemInfo(RVData.GetItem 190(i));
      for r := 0 to table.Rows.Count-1 do
        for c := 0 to table.Rows[r].Count-1 do
          if table.Cells 537[r,c]<>nil then
            ConvertRVToUnicode(table.Cells[r,c].GetRVData 575);
    end;
  end;
end;

procedure ConvertToUnicode(rv: TCustomRichView);
var i: Integer;
begin
  ConvertRVToUnicode(rv.RVData 151);
  for i := 0 to rv.Style.TextStyles.Count-1 do
    rv.Style.TextStyles[i].Unicode := True;
end;
```

Unicode in Delphi/C++Builder 2009 or newer

In the new versions of Delphi/C++Builder, the String type is Unicode by default.

Many properties and parameters in TRichView become Unicode, see "Unicode and ANSI Text in TRichView" above.

Default (initial) values of some properties are changed:

- Unicode 446 property of text style (from *False* to *True*);
- TRichView.RTFReadProperties 151.UnicodeMode 274 (from *rvruNoUnicode* to *rvruOnlyUnicode*);
- TRichView.Options 146 (*rvoAutoCopyUnicodeText* is included, *rvoAutoCopyText* is excluded).

When saving text styles (in RVF files or Delphi forms) in older versions of Delphi/C++Builder, only non-default value (*True*) of Unicode property of text style is saved. When saving text styles (in RVF files or Delphi forms) in Delphi/C++Builder 2009+, value of Unicode property is always saved, default or not. The main consequence is the following: when loading forms/RVF files with styles saved by older versions of Delphi/C++Builder in Delphi/C++Builder 2009+, Unicode property of all text styles become *True*. For RVF files, all text in text items is converted to Unicode automatically.

ANSI text may appear in document when reading RTF files, if TRichView.RTFReadProperties 151.UnicodeMode 274<>*rvruOnlyUnicode*. If you use projects converted from the older version of Delphi/C++Builder, check a value of this property.

Import and Export

Text Files

LoadText 212, LoadTextFromStream 213 load ANSI text files. When loading to Unicode style, they perform conversion from ANSI to Unicode.

LoadTextW 212, LoadTextFromStreamW 213 load Unicode text files. When loading to non-Unicode style, they perform conversion from Unicode to ANSI.

Code page used for conversion is based on Charset 434 property of the corresponding style (Charsets of Unicode styles are used only for conversion to/from ANSI).

Note: you can test file with the function

```
function RV_TestFileUnicode(const FileName: String): TRVUnicodeTestResult
```

defined in RVUni.pas.

Return values

- *rvutNo* – the file is not Unicode (odd size);
- *rvutYes* – the file is most likely Unicode (even size, Unicode byte-order characters at the start or #0 in text (first 500 bytes checked));
- *rvutProbably* – the file can contain Unicode (even size);
- *rvutEmpty* – the file is empty;
- *rvutError* – error opening the file.

You can also use WinAPI function **IsTextUnicode** performing more advanced tests.

SaveText 221 saves ANSI text file. Unicode strings are converted basing on Style.DefCodePage 388 property.

SaveTextW 221 saves Unicode text file. ANSI strings are converted basing on the corresponding Charsets.

RTF (Rich Text Format)

Methods for RTF saving are able to store Unicode.

Methods for RTF loading and inserting work depending on TRichView.RTFReadProperties.UnicodeMode 274.

HTML

SaveHTML*** 62 can save ANSI or Unicode (UTF-8) HTML files. In ANSI HTML files, Unicode characters are written as codes (&#NNNN;), so all Unicode characters are preserved, but file size is increased; so it's highly recommended to save HTML in UTF-8 encoding.

Selection, Search and The Clipboard

GetSelTextA [204] returns selection as an ANSI string. Unicode text is converted basing on Style.DefCodePage [388] property.

GetSelTextW [204] returns selection as a Unicode string. ANSI strings are converted basing on corresponding Charsets.

Text searching methods have versions allowing to search for ANSI and for Unicode string: TRichView.SearchTextA/SearchTextW [222], TRichViewEdit.SearchTextA/SearchTextW [336]. All methods can search both in ANSI and Unicode text items. When comparing ANSI text with Unicode text, SearchText methods use Style.DefCodePage [388] property, SearchText methods use text Charsets.

CopyTextA [178] copies selection as ANSI text. Unicode strings are converted basing on Style.DefCodePage [388] property.

CopyTextW [178] copies selection as Unicode. ANSI strings are converted basing on corresponding Charsets.

None: on NT-based systems (such as Windows XP), the Clipboard is able to convert Unicode text to ANSI text and vice versa. So, if you copy in one of these formats, both formats are available for pasting.

Copy [177] and CopyDef [177] are able to copy Unicode (Option [146]-rvoAutoCopyUnicodeText)

Editing Operations

If pasting text using Paste [328] method, and both ANSI and Unicode texts are available in Clipboard, then the choice is made depending on the current text style (Unicode or not).

PasteTextA [331] pastes ANSI text, PasteTextW [331] pastes Unicode text.

InsertTextFromFile [328]: the file must be ANSI (converted, if needed)

InsertOEMTextFromFile [321]: the file must be OEM (converted, if needed)

InsertTextFromFileW [328]: the file must be Unicode (converted, if needed)

InsertText [327], InsertStringTag [326] add Unicode string in Delphi/C++Builder 2009+ and ANSI string in older versions of Delphi/C++Builder.

InsertTextA [327], InsertStringATag [326] add ANSI string (converted, if needed)

InsertTextW [327], InsertStringWTag [326] add Unicode string (converted, if needed)

RVF (RichView Format [59])

Applications compiled with older versions of RichView (version less than 1.2) will not be able to load RVF files with Unicode.

RVF files will be loaded correctly even if Unicode flags in text styles are mismatched (saved with different RVStyle then loaded), conversions will be performed if required (for example, this conversion will occur when loading old RVF files in applications compiled in Delphi/C++Builder 2009+). There are two RVF Warnings [155]: *rvfwConvToUnicode* and *rvfwConvFromUnicode*, which indicate if any conversion took place.

TRichView v11 introduces a new change in RVF files allowing to store String properties as Unicode. RVF files saved in Delphi/C++Builder 2009+ are saved as *RVF version 1.3.1*, RVF files saved in the older versions of Delphi/C++Builder are saved as *RVF version 1.3*.

See also...

- TRVStyle.DefUnicodeStyle [389];
- Example how to load UTF-8 files [282].

1.1.6.2 Bidirectional text

Bidirectional mode specifies the reading order of text, order of items in lines, and placement of a vertical scrollbar.

The support for bidirectional text (Arabic or Hebrew) is turned off by default. In order to turn it on, set BiDiMode 137 property of TRichView either to *rvbdLeftToRight* or to *rvbdRightToLeft*.

BiDiMode of document can be overridden by BiDiMode of paragraph 450, BiDiMode of paragraph can be overridden by BiDiMode of text item 433.

Mixed right-to-left and left-to-right text is supported, even inside the same text item.

Attaching of adjacent characters (common for Arabic languages) is supported too, but only inside the same item (not between items).

Bullets and numbering of paragraphs take BiDiMode into account: for RTL (right-to-left) paragraphs, list markers are positioned to the right.

Tab stops support BiDiMode too: they are inverted (including tab alignments) for RTL paragraphs.

Differences in Windows API and Uniscribe

The results depends on the value of TextEngine 403:

- if Windows API is selected, in *rvbdUnspecified* mode the results are completely incorrect for RTL text (wrong selection drawing and caret positions); the results are unreliable in other modes (may depend on the chosen font);
- if Uniscribe is selected, the results are correct is all modes; in *rvbdUnspecified* mode, only rearrangement of items on lines is turned off.

Limitations

- BiDiMode works in Windows versions supporting right-to-left languages (for example, on Win2000 and newer with RTL support installed, see Control Panel, Regional and Language Options, tab Languages, Install files for complex scripts and right-to-left languages). In all other versions of Windows, right-to-left text will be displayed left-to-right.
- (if Windows API is used) BiDiMode mode for Unicode text works only under WinNT-based systems (WinNT 4, 2000, XP, 7, 8 and so on)

Demo project

Demos\Delphi\Assorted\International\RTL

Demos\CBuilder\Assorted\International\RTL

1.1.7 Spelling Check

1.1.7.1 Live spelling check

Live spelling check marks misspelled words with special underlines (wavy line of TRVStyle.LiveSpellingColor 397 color).

Spelling check is executed in background thread.

Starting and finishing spelling check

In TRichView 128:

Spelling check is started by calling StartLiveSpelling 237 method.

In TRichViewEdit 282:

Spelling check is started:

- by calling StartLiveSpelling[237] method or
- when user modifies document or when any editing-style method is called (if LiveSpellingMode[293] =rvlspOnChange)

Spelling check is stopped by ClearLiveSpellingResults[175] method. Clear[175] also stops checking.

Checking Words

TRichView does not have its own spelling check engine and dictionaries. You need to use third-party components.

Currently the following components are supported:

1. ***Addict 3 and 4*** by Addictive Software (shareware Delphi component)
2. ***ExpressSpellChecker*** by Developer Express Inc. (shareware Delphi component)
3. ***HunSpell*** (C++ library under GPL/LGPL/MPL tri-license)
4. ***ASpell*** by Kevin Atkinson (DLL, license: LGPL)
5. ***Polar SpellChecker Component by*** by Polar (shareware ActiveX)
6. EDSSpell 7.0 by ***OneDomain, Inc*** (shareware Delphi component, discontinued)
7. VSSpell 8.0 by ***ComponentOne LLC*** (shareware ActiveX, discontinued)
8. VSSpell 3.0 by Visual Components, Inc (shipped with older versions of Delphi, ActiveX)
9. ***LS Speller*** by Luzius Schneider (ISpell + MS Office (CSAPI) spell checkers, freeware)

Let me know if you are interested in supporting other spelling checkers.

Files required for these spell-checkers can be downloaded from Resources for TRichView web page.

The live spelling thread calls OnSpellingCheck[265] event for each word in the document.

Word is defined as a run of characters between delimiters[140]. Additionally, '&' is treated as a delimiter. Apostrophes are processed specially, see RichViewApostropheInWord[653] global variable.

Single quote character is processed specially: if it's inside word, it's treated as a part of word (for example "it's" or "I'll"). If it's to the left/right, it's treated as a delimiter.

URLs are ignored, see RVIsCustomURL[662] function.

Making Corrections

Call LiveSpellingValidateWord[208] if user added word in a custom dictionary or ignore list.

Call GetCurrentMisspelling[315] to get or to correct the current misspelled word. For example, you can use this method to implement popup menu:

1. when context menu is about to be shown, use GetCurrentMisspelling[315] to get misspelled word in the caret position and to generate suggestions;
2. when the user has chosen a suggestion, use GetCurrentMisspelling[315] again to select the misspelled word and use InsertText[327] to insert replacement.

Alternatively, misspelled word can be selected on the step 1 (before displaying popup menu).

Spelling Check and Document Editing

When calling a editing-style method, you do not need to worry about live spelling, everything is processed automatically (the method pauses the live spelling thread, makes changes, then resumes the live spelling thread from the proper position).

However, when calling any viewer-style method for modifying a document, you must be sure that live spelling is not running. If you call Clear[175] and then create a new document[43], everything is OK, because Clear stops live spelling check. But if you make changes in existing document, call ClearLiveSpellingResults[175] before these changes.

Memory Usage

Live spelling support adds 8 extra bytes per each item. If you do not use it, you can compile your application with RVDONOTUSELIVESPELL compiler define (see RV_Defs.inc file)

1.1.8 Important Technical Information

1.1.8.1 Viewer vs Editor

Methods of TRichView

TRichView 128 has the following 3 groups of methods for modifying documents:

1. appending items to the end of the document (may be from file or stream);
2. modifying item with the given index;
3. loading files and streams.

Examples of the first group are: AddNL 167, AddPictureEx 169, LoadRTF 209.

Examples of the second group are SetPictureInfo 234, SetItemText 232.

Examples of the third group are: LoadRVF 211, LoadRTFFromStream 210.

All these methods have the following characteristics:

- they do not require that the document is formatted (formatting is a preparing document for displaying)
- even if the document was formatted before calling such method, it is not formatted after.

You need to call Format 182 (or FormatTail 183) to display documents created or modified with such methods.

Table cells 567 also have such methods.

These methods do not pause a live spelling thread, and it may lead to errors. Before calling these methods, make sure that live spelling is not running (by calling Clear 175 or ClearLiveSpellingResults 175).

Methods of TRichViewEdit

TRichViewEdit 282 has the following groups of methods for modifying documents:

1. all methods of TRichView;
2. methods for inserting items in the position of caret (**Insert***** methods, such as InsertText 327, InsertPicture 322);
3. methods for modifying the item at the position of caret (**SetCurrent***** methods, such as SetCurrentPictureInfo 348, SetCurrentItemText 347);
4. extended set of methods for modifying item with the given index (**Set***Ed** methods, such as SetPictureInfoEd 354, SetItemTextEd 352).

Methods of the group 1 are referred as viewer-style methods, methods of the groups 2-4 are editor-style methods.

All these methods (except for the methods of the group 1, inherited from TRichView) have the following characteristics:

- document must be formatted before calling such methods;
- these methods automatically reformat the affected area of documents;
- if the document is read-only 294 they do nothing; they also respect protection of text 439 and paragraphs 453;
- they call OnChange 357 event;
- they can be safely called while a live spelling is running;
- modifications can be undone and redone 54.

In an editor, the main application of the methods of the group 1 is generating new documents from scratch (after

calling Clear[175]).

The main application of other methods is performing user commands. They should not be used for generation of new document (because each such method updates documents, it can cause undesirable flicker, and they are much slower).

Important: You must not mix methods of group 1 and methods of groups 2-4! Methods inherited from TRichView do not update undo and redo buffers, and results of undo and redo commands will be unpredictable!

The only applications of methods of the group 1 is generating a new document, in the sequence:

1. Clear[175]; (this method also clears undo and redo buffers);
2. One or several calls of methods of the group 1;
3. Format[182] (after the document has been formatted, you can apply methods of the groups 2-4)

See also: Undo in tables[121]

1.1.8.2 Controls - Documents - Items

There are three main families of classes.

1. VCL controls, descendants of TCustomRichView.

See the diagram of class hierarchy[27].

The main visual controls are: TRichView[128], TRichViewEdit[282], TDBRichView[373], TDBRichViewEdit[377].

2. Documents, objects containing items, descendants of TCustomRVFormattedData[608].

Documents are objects, but not components.

Each RichView or RichViewEdit has such document accessible as RVData[151] property.

They have the most of methods of RichView controls. For example, calling

```
MyRichView.AddNL('text',0,0)
```

is equivalent to

```
MyRichView.RVData.AddNL('text',0,0)
```

Tables[528] have cells[567]. Cells are also documents containing items, cells are descendants of TCustomRVFormattedData.

3. Items of documents, descendants of TCustomRVItemInfo[526].

Items are objects, but not components.

Items are usually hidden from programmers. They can be added and modified by methods of controls or document objects. The only exception is tables (TRVTableItemInfo[528]).

Documents contain items, some items (tables) can contain documents (cells).

There is an important case: cell editing. When editing cell, RichViewEdit creates a special editor (also TRichViewEdit) on the top of the cell. While editing, all items in the cell are moved in this editor. When editing is finished, items are moved back to the cell.

At any time, object containing cell items is accessible as cell.GetRVData[575] (if cell is not edited, it is the cell itself; if edited, it is RVData[151] of its inplace editor).

Example 1

This example shows that the same procedure can be applied to the main document and table cells.

This example converts all text to upper case

```
procedure AllUpperCase(RVData: TCustomRVData[608]);
var i,r,c: Integer;
    s: String;
```

```
    table: TRVTableItemInfo[528];
begin
for i := 0 to RVData.ItemCount[144]-1 do
  if RVData.GetItemStyle[194](i)>=0 then begin // this is a text item
    s := RVData.GetItemText[195](i);
    s := AnsiUpperCase(s);
    RVData.SetItemText[232](i,s);
    end
  else if RVData.GetItemStyle[194](i)=rvsTable[663] then begin
    table := TRVTableItemInfo[528](RVData.GetItem[190](i));
    for r := 0 to table.RowCount[543]-1 do
      for c := 0 to table.ColCount[538]-1 do
        if table.Cells[537][r,c]<>nil then
          AllUpperCase(table.Cells[537][r,c].GetRVData[575]);
  end;
end;
```

Call:

```
AllUpperCase(MyRichView.RVData);
MyRichView.Format[182];
```

Example 2

The same task, using different methods

```
// This procedure will be called for each RichView item
procedure TForm1.EnumItemsProc(RVData: TCustomRVData[608];
  ItemNo: Integer;  var UserData1: Integer;
  const UserData2: String; var ContinueEnum: Boolean);
var s: String;
begin
  if RVData.GetItemStyle[194](ItemNo)>=0 then begin
    s := RVData.GetItemText[195](i);
    s := AnsiUpperCase(s);
    RVData.SetItemText[232](ItemNo,s);
  end;
  ContinueEnum := True;
end;
```

Call:

```
var v: Integer;

v := 0;
// RVData.EnumItems calls the specified procedure (EnumItems)
// for each items in RVData and its subdocuments (table cells)
// The second and the third parameters are user defined data,
// they will be passed in the procedure as UserData1 and UserData2
MyRichView.RVData[151].EnumItems(EnumItemsProc, v, '');
MyRichView.Format[182];
```

The advantage of using RVData.EnumItems is more clear and compact code.

The disadvantage: you cannot use it if you need to add or delete items.

1.1.8.3 Valid Documents

There are some limitations on document structure.

Internally, each document item[85] has properties:

- index of paragraph style
- "this item starts a paragraph" flag
- "this item starts a line inside a paragraph" flag

Not all combinations are acceptable.

All operations in editor[282] always create a valid document (providing that it was valid before the operation).

But some operations in viewer[128] can create invalid documents (especially DeleteItems[179])

Rules of Valid Documents:

1. All items in paragraphs must have the same index of paragraph style (in the most of cases, RichView provides this. It's possible to create invalid documents by incorrect using of DeleteItems[179] and SetAddParagraphMode[225])
2. The first item (in document or table cell) must start a paragraph.
3. Full-line items (*breaks*[93] and tables[98]) must start a new paragraph.
4. Items after full-line items must start a new paragraph.
5. Table cells must contain at least one item (by default they contain one empty text item).
6. List markers[99] must start a new paragraph.
7. Paragraph after marker must contain at least one item (may be an empty text item). Documents where next item after marker starts a new paragraph (or a line) are incorrect
8. Empty text items can be only in empty paragraphs (the only item in paragraph, or the only item after list marker)

RichViewActions includes RVNormalize unit containing NormalizeRichView procedure. This procedure can fix the most of document problems.

1.1.8.4 Units of Measurement

Glossary

Pica (pc) – typographic unit of measure corresponding to 1/6 of inch. The pica contains 12 points.

Point (pt) – typographic unit of measure corresponding to 1/72 of inch. This is a measuring unit for font size[441].

Twip (tw, "twentieth of a point") is defined as 1/20 of point. It equals to 1/1440 of inch.

Pixel (px) – a single point in a raster image or device. To convert inch (or other absolute unit of measurement) to pixels, a resolution (referred as "pixels per inch" or "dots per inch") of this device must be known. For some devices (like printers), this value is precise. For other devices (like computer displays), this value is logical and can be changed by the user. For screens, a typical "pixels per inch" value is 96 (accessible as Screen.PixelsPerInch).

In TRichView, properties can be measured in **"screen pixels"**. It means that on the screen, this length includes the specified number of pixels; but when printing, it is recalculated according to the printer resolution as:

value * (printer resolution) / (screen resolution)

Screen resolution can be read from Screen.PixelsPerInch, or it can be overridden in RichViewPixelsPerInch[655] global variable.

Properties measured in points

Font size (Size 441 property of text styles 404)

Properties measured in screen pixels

The following properties of TCustomRichView 128 are measured in screen pixels:

- LeftMargin 145, RightMargin 149, TopMargin 158, BottomMargin 137 – margins;
- MaxTextWidth 146 – maximal width for word wrapping;
- MinTextWidth 146 – minimal width for word wrapping (minimal width of scrollable area, not including margins).

Properties measured in TRVUnits 648

Properties of the type TRVLength 636 are measured in TRVUnits 648. They are values of a floating point type (Extended) that can represent lengths measured in inches, centimeters, millimeters, picas, points, or screen pixels.

In TRichView, this type is used for:

- properties of TRichView 128.DocParameters 140; they are measured in TRichView 128.DocParameters 140.Units 280.
- properties of TRVPrint 479; they are measured in TRVPrint 479.Units 487.

In RichViewActions, this type is used to define units of values displayed in dialog windows. They are measured in TRVAControlPanel.UnitsDisplay.

In ScaleRichView, this type is used for properties of TSRichViewEdit.PageProperty and TSRichViewEdit.ViewProperty. They are measured in TSRichViewEdit.UnitsProgram.

Properties measured in TRVStyleUnits 644

Properties of the type TRVStyleLength 644 are measured in TRVStyleUnits 644. They are values of an integer type that can represent lengths measured in twips or screen pixels.

The property TRVStyle 385.Units 404 defines units for properties (and sub-properties) of this TRVStyle component as well as for properties of items 85 in documents loaded in linked 156 TRichView controls.

Properties of TRVStyle 385 measured in TRVStyleUnits 644

The following properties are measured in TRVStyleUnits 644:

- DefTabWidth 388;
- CharSpacing 435 of text style 444;
- FirstIndent 451, LeftIndent 451, RightIndent 454, SpaceBefore 455, SpaceAfter 454 of paragraph style 456;
- Position 471 of item in a tab stop collection 455 of paragraph style;
- Left, Right, Top, Bottom of background padding rectangle (Background 450.BorderOffsets 466) of paragraph style;
- Left, Right, Top, Bottom of border padding rectangle (Border 451.BorderOffsets 466) of paragraph style, as well as border's Width 468 and InternalWidth 468;
- MarkerIndent 478, FirstIndent 473, LeftIndent 476 in list levels 459 of list style 459.

Item 85 properties measured in TRVStyleUnits 644

The following items properties 627 are measured in TRVStyleUnits 644:

- *rvepSpacing, rvepImageWidth, rvepImageHeight, rvepVShift* (the latter only if *rvepVShiftAbs* is nonzero)

The following table 98 properties are measured in TRVStyleUnits 644:

- BorderWidth 534, CellBorderWidth 535;
- BorderHSpacing 532, BorderVSpacing 533, CellHSpacing 536, CellVSpacing 537;
- CellHPadding 536, CellVPadding 537, CellPadding 536;
- HRuleWidth 540, VRuleWidth 545;
- BestWidth 531 (only if value is positive).

The following table cell 567 properties are measured in TRVStyleUnits 644:

- BestHeight 568 (only if value is positive);
- BestWidth 569 (only if value is positive).

A width of break 93 (horizontal line) is measured in TRVStyleUnits 644 as well.

The following label item 101 properties are measured in TRVStyleUnits 644:

- MinWidth 580.

The following sidenote 105 properties are measured in TRVStyleUnits 644:

- BoxProperties 592.Width 595 and Height 594 (if WidthType and HeightType are "absolute");
- BoxPosition 592.HorizontalOffset 597 and VerticalOffset 600 (only if HorizontalPositionKind and VerticalPositionKind are "absolute").

In RichViewActions, this type is used for TRVAControlPanel.UnitsProgram (units of measurement for properties of several actions).

Import and export

In RTF (Rich Text Format) files, native units are twips, so all properties are saved as twips. When loading, they are converted to TRVStyle 385.Units 404.

In HTML, native units are pixels, so all properties are saved in pixels. In HTML CSS, these properties are saved either as integer values in pixels (if TRVStyle 385.Units 404=*rvstuPixels*), or as fractional values in points (if TRVStyle 385.Units 404=*rvstuTwips*).

In RVF 59 (RichView Format) files, values are saved as TRVStyle 385.Units 404. When loading, they are either converted to TRVStyle 385.Units 404, or vice versa, TRVStyle 385.Units 404 is changed. See "Units of measurement" in RVF Overview 59. Alternatively, TRichView can be converted to units from RVF file, if *rvfoCanChangeUnits* is included in RVFOptions 152.

1.1.8.5 Custom drawing

There are the following events for custom drawing.

TCustomRichView 128

- OnPaint 247 – drawing over RichView document.

TRVPrint 479

- OnPagePrepaint 495 – drawing on page under the document;
- OnPagePostpaint 495 – drawing on page over the document;
- OnPrintComponent 519 – printing components.

TRVStyle 385

- OnDrawCheckpoint 411 – drawing checkpoint;
- OnDrawPageBreak 413 – drawing page break;
- OnDrawParaBack 414 – drawing background of paragraph;
- OnDrawTextBack 416 – drawing background of text;
- OnApplyStyle 410 – applying text style to Canvas;
- OnApplyStyleColor 411 – applying the text style color to Canvas;
- OnDrawStyleText 415 – drawing text of the given style;
- OnStyleHoverSensitive 417 – returns "should component be repainted when user moves the mouse to a hypertext of the given style?"

TRVTableItemInfo 528

- OnDrawBorder 565 – drawing table and cell borders.

1.1.8.6 Cursors

Image	Constant	Value	Used by	Unit
 *	*crJump*	101	TRVStyle 385 component: ▪ TRVStyle.JumpCursor 394; ▪ TFontInfo.JumpCursor 438.	**RVStyle.pas**
	*crRVZoomIn*	102	TRVPrintPreview 384 component (ZoomInCursor 523)	**CRVPP.pas**
	*crRVZoomOut*	103	TRVPrintPreview 384 component (ZoomOutCursor 524)	**CRVPP.pas**
	*crRVSelectRow*	104	TRVTableItemInfo 528	**RVTable.pas**
	*crRVSelectCol*	105	TRVTableItemInfo 528	**RVTable.pas**
	*crRVFlipArrow*	106	TRVStyle 385 component (LineSelectCursor 394)	**RVStyle.pas**
	*crRVLayingIBeam*	107	TRichViewEdit 282	**RVStyle.pas**

* in Windows2000+, RichView uses the system default hypertext cursor instead (since version 1.4)

1.1.8.7 RVF Specification

This topic contains thorough information about RichView file Format.

For the most of applications you need not to know about internals of RVF, all necessary information is in RVF Overview 59.

Syntax

{x} – zero or more repetitions of item x;

{x}+ – one or more repetitions of item x;

[x] – item x is optional;

xy – item x is followed by item y;

x | y – item x or item y;

'c' – a literal;

<n> – a nonterminal.

Data Types in RVF

<int> – an integer number, ['-']{<digit>}+;

<digit> ::= '0' | ... | '9';

Integer number is either finished by ' ' (space character) or it is followed by the end of line; for the simplification, this space character will be omitted (i.e. it will be included in <int>).

<crlf> – end of line characters; usually it is a CR+LF, CR or LF characters; before binary data or Unicode it is required to use both CR and LF characters (#13#10).

<string> – string of ANSI characters (without CR, LF and #0 characters)

<Unicode string> – string of Unicode characters. Last character in string must be a paragraph separator (code $2029).

<special string> – string containing ANSI text if RVF version is 1.3 or lower, and UTF-8 text if RVF version is 1.3.1. In this string, character #1 represents character #13, character #2 represents character #10.

<quoted string> – <special string> enclosed in double quotes; double quotes inside the string are doubled.

RVF Structure

<RVF> ::= [<Version Info>]{<RVF Record>}

Version Info

<Version Info> := <Version Record Type><Version><SubVersion>[<SubSubVersion>]<crlf>

<Version Record Type> ::= `'-8'`

<Version> ::= <int>

<SubVersion> ::= <int>

<SubSubVersion> ::= <int>

Currently, <Version> = 1, <SubVersion> = 3. If there is no <Version Info> in RVF, reader implies version 1.0 (no <Item Options> in <RVF Item>)

<SubSubVersion> = 1 for Delphi 2009+, and omitted for older versions of Delphi.

RVF Record Structure

RVF file consists of *records*. The most of records represent items (one record = one item), but there are some special record types.

<RVF Record> ::= <Record Header><crlf>({<string><crlf>}|{<Unicode sting>})[<Binary Data>]

<Record Header> ::= <Record Type><Strings Count><Paragraph><Item Options><Query><Tag><Rest of Header>

<Record Type> ::= <int>

<Strings Count> ::= <int>

<Paragraph> ::= <int>

<Item Options>::=<int>

<Query> ::= `'0'` | `'1'` | `'2'` | `'3'`

<Tag> ::= <quoted string>

<Record Type> is a type of <RVF Record>

<Record Type>	Constant	Meaning
>=0	—	text 88; <Item Type> is an index in the collection of styles (RVStyle.TextStyles 404)
-1,-3..-6, -10..-12, -60, ...	*rvs*** 663 constants	non-text item 85
-2	*rvsCheckpoint*	invisible label ("checkpoint" 30)
-7	*rvsBack*	background (image, color)
-8	*rvsVersionInfo*	RVF version (only in <Version Info>)

-9	*rvsDocProperty*	text, paragraph, list styles,layout information, DocProperties [141], DocParameters [140]

<Strings Count> is a number of lines between this <Record Header> and the next <Record Header> (or the end of file). If RVF reader does not understand this record type it should skip this number of lines. <Binary Data> (if present) is counted as one of strings after the header..

<Paragraph> is an index in the collection of paragraph styles (RVStyle.ParaStyles [398]) (if this item starts a paragraph) or -1 (if the record represents item, and this item continues paragraph).

<Item Options> is a set of bits, converted to integer.

Bit Number	Meaning
0	Set if the item continues paragraph (<Paragraph>=-1); it's a redundant data;
1	Page break. Set if this item must start a new page (<Paragraph><>-1)
2	Set if the item should be displayed from the new line, but it does not start a new paragraph (in this case <Paragraph><>-1, but should be equal to the paragraph style of the previous non "-1" item)
3	Set if it is a text item containing Unicode.

<Tag>

<Tag> is a string associated with this item. RichView saves this string in double quotes. Empty tags are saved as '0' (without quotes). For Delphi 2009+, tag strings are saved as UTF-8, for older versions of Delphi they are saved as ANSI strings. On saving, #13 and #10 are changed to #1 and #2, on loading, there are restored back.

See also *tags* [34].

<Query> can be a digit in the range 0..3

<Query>	Meaning
0	The item content is saved in a text format (*rvfoSaveBinary* is excluded from RichView.RVFOptions [152])
1	The item is requested from program on loading, see below.
2	The item content is saved in a binary format (rvfoSaveBinary is in RichView.RVFOptions [152]).
3	Special value for saving Unicode text as hexadecimal string

If <Query>=1, then the next line after the header is a name of item (picture or control).

Data is requested in events: OnRVFPictureNeeded [252], OnRVFControlNeeded [251].

The saving mode (to save full information about images or controls / to save only name and request it when reading) is determined by RichView.RVFOptions [152] (for the request mode, *rvfoSavePicturesBody* and/or *rvfoSaveControlsBody* options should be cleared). You need to create control or image in this event and assign it to **gr** or **ctrl** parameters. For example:

```
procedure TMyForm.MyRichViewRVFControlNeeded(
  Sender: TCustomRichView; Name: String;
  Tag: TRVTag [645]; var ctrl: TControl);
begin
  ctrl := TButton.Create(nil);
  TButton(ctrl).Caption := 'Demo';
  TButton(ctrl).OnClick := MyClickProcedure;
```

```
end;
```

<Rest of Header> varies for different item types. Some new parameters can be added in future to <Rest of Header>. RVF reader should ignore all parameters in header after last parameter it can understand.

If <Query> is equal to 1 then <Rest of Header> is empty.

The next section is for <Query>=0 | 2 | 3.

RVF Records

Checkpoints

<Record Type> = *rvsCheckpoint*

<Paragraph> is ignored.

<String Count> = 1

<Rest of Header> ::= <Raise Event>

<Raise Event> ::= `'0'` | `'1'`

If Raise Event is equal to `'1'` then OnCheckPointVisible[238] event occurs when the checkpoint appears in visible area of RichView

Strings after the header:

- <Checkpoint name>

<Checkpoint name> ::= <special string>

"Checkpoint" is not an item, but it's convenient to save it like an item. "Checkpoints" are saved before the associated items.

Text[88] (Items)

<Record Type> >= 0

<Rest of Header> := [<Hint>]

<Hint> := <special string>, value of rvespHint[630] property.

<Strings Count> is a number of paragraphs in this item. All paragraphs will be of the given text and paragraph style. If <Paragraph>=-1 then this item continues the paragraph of the previous item.

Strings after the header are text lines (paragraphs). Stings can be in Unicode encoding (determined by <Item Options>). If <Query>=0, Unicode is saved as UTF-16 (WideString). If <Query>=3, Unicode is saved as hexadecimal string

Break[93] (Item)

<Record Type> = *rvsBreak*

<Paragraph> is ignored

<Rest of Header> ::= <Break Color><Break Style><Break Width>

<Break Color> ::= <int>

<Break Style> ::= <int>

<Break Width> ::= <int>

<Break Color> is a color of horizontal line; if Break Color=*clNone* then line will be of color of the 0th text style.

<Break Style> is TRVBreakStyle[624] value converted to integer.

<Break Width> is a width of line.

<String Count> >= 0.

Picture 90 (Item)

<Record Type> = *rvsPicture*

<String Count> >= 3

<Rest of Header> = <Vertical Align>

<Vertical Align> ::= <int>

<Vertical Align> is a vertical alignment of picture (TRVVAlign 648 value converted to integer)

Strings after the header:

- <Picture Name>
- <Picture Delphi Type>
- <Picture Data>

<Picture Name> ::= <string>, it is an item text.

<Picture Delphi Type> is a name of picture class (such as `'TBitmap', 'TIcon', 'TMetafile', 'TJPEGImage'`).

<Picture Data> can be in text or binary format.

In text format, each byte is represented as two hexadecimal digits, so all data is encoded in one string.

In binary format, <crlf> after <Picture Delphi Type> must be CR+LF (both characters). Binary data consists of <Data Size> and <Data>. Data Size is a binary 4-byte number, the number of bytes in <Data>.

Note 1: In order to load picture of the given class from RVF, its class must be registered with **RegisterClasses** procedure. You need not to register TBitmap, TIcon, TMetafile, because they are registered by RichView. For Delphi 3+ RichView also registers TJpegImage (if the conditional define RVDONOTUSEJPEGIMAGE is not defined) In order to use third party graphics formats (descendants of TGraphic) in RVF you need to register them yourself with **RegisterClasses** procedure.

Note 2: Behavior of RichView when loading unregistered graphics format is defined in RVFOptions 152 property. If *rvfoIgnoreUnknownPicFmt* option is defined then this picture will be just skipped. If this option is not defined then the function returns False (reading failed). In both cases RVFWarnings 155 property will contain *rvfwUnknownPicFmt*.

Control 94 (Item)

<Record Type> = *rvsComponent*

See the previous section, adjusted for: *rvfoIgnoreUnknownCtrls*, *rvfwUnknownCtrls*.

Bullet 96 (Item)

<Record Type> = *rvsBullet*

<String Count> >= 1

<Rest of Header> = <ImageList#><ImageIndex>

<ImageList#> ::= <int>

<ImageIndex> ::= <int>

Strings after the header:

- <Bullet Name>

<Bullet Name> ::= <string>, it is an item text.

<ImageList#> is the Tag property of the corresponding ImageList (do not confuse it with RichView item tags! It is TImageList.Tag property). On reading, OnRVFImageListNeeded 252 occurs. You need to assign your imagelist to **il** parameter. Otherwise, the bullet will be skipped. The behavior of RichView when loading too large image index is defined in RVFOptions 152 property. If *rvfoConvLargeImageIdxToZero* option is defined then index is set to 0. Otherwise the function returns False (reading failed). In both cases RVFWarnings 155 property will contain *rvfwConvLargeImageIdx*.

Hotspot[97] (Item)

<Record Type> = *rvsHotspot*

<String Count> >= 1

<Rest of Header> = <ImageList#><ImageIndex><HotImageIndex>

<ImageList#> ::= <int>

<ImageIndex> ::= <int>

<HotImageIndex> ::= <int>

Strings after the header:

- <Hotspot Name>

<Hotspot Name> ::= <string>, it is an item text.

See the previous section for details.

Collections of styles, layout, DocProperties[141]

<Record Type> = *rvsDocProperty*

<Paragraph> is ignored

<Query> can be 0 (data will be saved as a hexadecimal text) or 2 (in a binary form)

<Rest of Header> ::= `'1' | '2' | '3' | '4' | '5' | '6'`

<Rest of Header>	Meaning
1	Collection of text styles
2	Collection of paragraph styles
3	Layout (margins, BiDiMode[137], MaxTextWidth[146], MinTextWidth[146]) of TRichView
4	Collection of list styles
5	DocProperties[141]
6	Special information, written by TRVReportHelper[503] when saving page
7	DocParameters[140]
8	Subdocument (header and footer for TSRichViewEdit)
9	RVStyle.Units[404]

<String Count> = 2 for 1..4 and 6. The first line contains name of TRVStyle object for 1,2,4, or empty for 3,6. Hexadecimal text or binary data follow next.

When saving DocProperties, <String Count>=DocProperties.Count. Each item in DocProperties is saved as <special string>.

DocParameters are saved as lines like property=value (only for properties with non-default values).

Background

<Record Type> = *rvsBackground*

This item is similar to *rvsPicture*, but has differences:

- <Rest of Header> is a background color and style
- there is no <Picture Name> line;

- <Picture Delphi Type> is always TBitmap
- <Query>=1 is not supported

<String Count> = 2 | 0

<Paragraph> and <Tag> are ignored (0).

<Rest of Header>=<Background Style><Color>

<Background Style>::= <int> (TRVBackgroundStyle[136] value converted to integer)

<Color>::=<int> (background color)

If background style is *bsNoBitmap* (or background bitmap is empty), there are no strings after header (and <Query>=0).

Strings after the header:

- <Picture Delphi Type>
- <Picture Data>

List Markers[99] (Item)

<Record Type> = *rvsMarker*

<String Count> >= 0

<Rest of Header> ::= <List Index> <Level> <Start From> <Reset>

All items in <Rest of Header> are <int> (<Reset> is ord(boolean value))

Label[101] (Item)

<Record Type> = *rvsLabel*

<String Count> >= 6

Strings after the header:

- <Text>
- <Text Style>
- <Min Width>
- <Alignment>
- `'protect'|'no protect'`
- <Item Name>

<Text> ::= <special string>, it is Text[581] property. <Text Style> ::= <int>, it is TextStyleNo[581] property. <Min Width> ::= <int>, it is MinWidth[580] property. <Alignment> ::= <int>, it is ord(Alignment[580]). The next line depends on ProtectTextStyleNo[581]. <Item Name> ::= <string>.

Sequence[102] (Item)

<Record Type> = *rvsSequence*

<String Count> >= 10

Strings after the header:

- <Sequence Id>
- <Numbering Type>
- <Reset>
- <Start From>
- <Text Style>
- <Min Width>
- <Alignment>
- `'protect'|'no protect'`
- <Item Name>
- <Formatting String>

<Sequence Id> ::= <special string>, it is SeqName[585] property. <Numbering Type> ::= <int>, it is NumberType

584 property. <Reset> ::= <int>, it is ord(Reset 584). <Start From> ::= <int>, it is Reset 585 property. <Text Style>. The next five lines have the same meaning as for labels. <Formatting String> ::= <special string>, it is FormatString 584 property.

Footnote 104 and EndNote 103 (Item)

<Record Type> = *rvsFootnote* | *rvsEndnote*

<String Count> >= 8

Strings after the header:

- <Text Style>
- <Min Width>
- <Alignment>
- `'protect'|'no protect'`
- <Reset>
- <Start From>
- <Item Name>
- <Document>

The first seven lines have the same meaning as in sequences.

<Document> can be in text or binary format. It is Document 586 saved in RVF format.

In text format, each byte is represented as two hexadecimal digits, so all data is encoded in one string.

In binary format, <crlf> after <Item Name> must be CR+LF (both characters). Binary data consist of <Data Size> and <Data>. Data Size is a binary 4-byte number, the number of bytes in <Data>.

Reference to Footnote or Endnote 108 (Item)

<Record Type> = *rvsNoteReference*

<String Count> >= 5

Strings after the header:

- <Text Style>
- <Min Width>
- <Alignment>
- `'protect'|'no protect'`
- <Item Name>

Meanings of these lines are the same as for labels.

Extra Item Properties

Some items may contain extra integer 627 and string 630 properties. They are saved as lines after the header (one line per property) in the form:

```
name=value
```

If the property has the default value, it is not saved. The list of names can be found in RVItem.pas, arrays RVFExtraItemIntPropNames and RVFExtraItemStrPropNames.

Since TRichView v11, values are saved as <special string>

This specifications does not cover a case of saving style names instead of style indices (this feature is deprecated)

Major Changes in RVF Format

RVF Version 1.2

Older programs are not able to read new RVF files, but programs compiled with new versions of RichView can read old RVF files.

The most significant change is adding <Item Options> in <RVF Item>.

There are two new record types: -7 (background) and -8 (version information)

RVF reader reads version number from version information. If there is no version information item, reader understand the file as RVF version 1.0

RVF version 1.3 (current version for Delphi 2007 or older):

- allows saving space characters in tag strings (for Delphi 3+); tags strings are saved in double quotes;
- in text mode, Unicode text is saved as a hexadecimal string (each byte is saved as two characters – hexadecimal digits)

RVF version 1.3.1 (current version for Delphi 2009 or newer):

- DocProperties 141, extra string properties 630, tag strings (if *rvoTagsArePChars* in Options 146; note: since TRichView v13.2 tags are always strings), style names (if *rvfoUseStyleNames* in RVFOptions 152) are saved as UTF-8 (with CR and LF characters changed to #1 and #2)

More Detailed Information about Changes in RVF

Changes in RichView version 1.3:

RVF now can contain Unicode text (for text items). Older versions will be unable to load files with Unicode. Version number in RVF header was not changed, because there are no changes in the specification (additions only).

Note: Unicode is used only for saving Unicode text items, so file contains a mix of ANSI and Unicode text. Such files must be considered as binary, do not edit them in text editors.

Changes in RichView version 1.4:

Added: tables (rvsTable). This is not a change in RVF, but an addition.

Tables are saved in binary or hexadecimal text form, similar to pictures or controls.

Changes in RichView version 1.5:

RVF now can contain text and paragraph styles (rvsDocProperty). This is not a change in RVF, but addition.

Styles are saved in binary or hexadecimal text form, similar to pictures or controls.

Changes in RichView version 1.7:

RVF now can contain layout information and collection of list styles (rvsDocProperty). This is not a change in RVF, but addition.

Changes in RichView version 1.8

RVF version number is changed to 1.3

Changes in RichView version 11

RVF version number is changed to 1.3.1 for Delphi 2009 and newer. Values of extra item properties are saved as <special string>, thus allowing them to contain line break characters.

Changes in RichView version 13

Saving measuring units. Sizes can be in twips.

1.2 Overview of Item Types

Item types

- RichView Item Types 86
- Text Items 88
- Tabulators 89

- Pictures 90
- *Hot-Pictures* 93
- *Breaks* 93
- Controls 94
- *Bullets* 96 (images from image-lists)
- *Hotspots* 97
- Tables 98
- List markers 99 (paragraph bullets & numbering)
- Labels 101 (non-text item looking like text)
- Numbered sequences 102
- Endnotes 103
- Footnotes 104
- Sidenotes 105 (notes displayed in floating boxes)
- Text Boxes 107 (floating boxes)
- References to parent notes 108
- "Page number" fields 109
- "Page count" fields 110
- Custom Item Types 111

1.2.1 Overview

Items

RichView documents consist of items. Items are indexed from 0 to ItemCount 144-1. Some items (tables 111) contain subdocuments (cells) which in their order contain items.
Items of subdocuments are not items of the main document, the are not included in the item range from 0 to ItemCount 144-1.

Item Types

Type ("style") of the given item is returned by the method GetItemStyle 194 (in editor, a style of the item at the position of caret is returned by CurItemStyle 289).

If the returned value is 0 or positive, this is a text item. In this case, the returned value defines a style of this text: it is an index in the TextStyles 404 collection of the the linked 156 TRVStyle 385 component.

If the returned value is negative, this is a constant identifying the item class.

The standard RichView item types are:

- text 88
- tabulators 89
- pictures 90
- *hot-pictures* 93 (pictures with hyperlink)
- *breaks* 93 (horizontal lines)
- controls 94 (any Delphi/C++Builder control)
- *bullets* 96 (images from image-list)
- *hotspots* 97 (bullets with hyperlink)
- tables 98
- list markers 99 (paragraph bullets & numbering)
- labels 101 (non-text item looking like text)
- numbered sequences 102
- endnotes 103
- footnotes 104
- references to parent footnotes and endnotes 108.

Below is the overview of properties which are common for all item types.

Text

Each item has associated text. For text items, this text is displayed. For non-text items, this text is not used by the component itself; you can store any information in it. Text must not contain characters: CR (#13), LF (#10), #0.

For text-items, text can be in traditional (ANSI) or in Unicode [64] encoding. ANSI text must not contain characters (CR (#13), LF (#10), #0, TAB (#9), FF (12).

See GetItemText [195], SetItemText [232] methods of TRichView and related methods of TRichViewEdit.

Tag

Tag is an additional string value associated with each item (Unicode for Delphi/C++Builder 2009+, ANSI for older versions).

The default value of tags is '' (empty string) .

For example, you can use tags to store targets of hyperlinks.

See also: Tags in RichView [34]

Checkpoint

Checkpoint marks the item and can be used to find this item quickly. *Checkpoints* are exported in RTF as bookmarks, in HTML as anchors.

See also: Checkpoints in RichView [30]

Pagebreak-before Flag

If this flag is set, this item starts a new page when printing.

See PageBreaksBeforeItems [149].

Paragraph Attributes

Items are organized in paragraphs. Paragraph attributes are defined in the ParaStyles [398] collection of the linked [156] TRVStyle [385] component.

GetItemPara [194] returns an index of paragraph style for the given item. This method returns the same value for all items in the same paragraph.

IsParaStart [208] returns True, if the given item starts a new paragraph.

IsFromNewLine [207] returns True, if this item starts a new paragraph or a new line in this paragraph.

Vertical Alignment

These properties can be applied only to non-text item types which are positioned like a character of text. They cannot be applied to items occupying a full line (such as tables and breaks).

By default, bottom of item is aligned to the base line of text (items of some types can be aligned differently, see TRVVAlign [648]).

You can move item up or down by the specified number of pixels/twips or percent. See the following extra item properties [627]: *rvepVShift*, *rvepVShiftAbs*.

For text items, see VShift [443] property of text style.

Spacing

You can define spacing around the item. See the following extra item properties [627]: *rvepSpacing*

Coordinates

GetItemCoords[191] returns coordinates of the top left corner of the given item relative to the beginning of the document.

GetItemClientCoords[191] returns coordinates of the left top corner of the given item relative to the top left corner of RichView window.

Protection

Text have several option of protection, see Protection[439] properties of text style.

You can protect non-text items from deletion. See the following extra item integer properties[627]: *rvepDeleteProtect*.

Visibility

Items can be hidden[42]. Hidden items are displayed only if *rvoShowHiddenText* is included in the Options[146] property of TRichView[128] control.

Text items are hidden, if *rvteoHidden* is included in the Options[438] property of their style[444].

Non-text items are hidden, if a non-zero value is assigned to *rvepHidden* extra item integer property[627].

Popup Hints

The component can show a hint message when the mouse pointer is above the item.

See the following extra item string properties[630]: *rvespHint*.

Hint message can be dynamically customized in OnItemHint[246] event.

Events

OnItemAction[244] occurs when item is inserted, deleted, moved to undo list, or when item text is modified.

OnSaveItemToFile[261] occurs when saving item in HTML, RTF or text file or stream. This is a "low level" event. It allows you to change how the item is saved.

1.2.2 Text Items

Text Item

Text item displays text in traditional (ANSI) or Unicode[64] encoding.

Style[194] of this item type: >=0 (index in the TextStyles[404] collection of the linked[156] TRVStyle[385]).

Item of this collection (TFontInfo[444] object) completely defines all properties of this text.

Main properties of TFontInfo[444]:

- Unicode[446] - determines if this text is Unicode[64] or not;
- Jump[445] - determines if this text is hypertext[35] or not;
- properties defining font, colors, background, hypertext cursors, protection, bi-di mode[69], and others.

Text item cannot contain:

- CR and LF (#13 and #10) characters;
- TAB (#09) characters (they are inserted as tabulators[89] or replaced with several space characters, depending on SpacesInTab[402]);
- #0 characters;
- FF (#12) characters (see PageBreaksBeforeItems[149] property of TCustomRichView).

Methods of TCustomRichView [128]

The following viewer-style methods add item of this type to the end of the document:

- single item: AddNLTag [167] (and shortcuts AddNL, AddTag, Add [167]), AddFmt [163]
- multiple items: AddTextNL [171];
- multiple items, can be used in Unicode applications: AddTextNLA, AddTextNLW [171];

The following viewer-style methods change text of the given item

- SetItemText -A -W [232]

The following methods return text of the given item:

- GetItemText -A -W [195]

Appending text from file/stream:

- LoadText, LoadTextW [212],
- LoadTextFromStream, LoadTextFromStreamW [213]

Methods of TCustomRichViewEdit [282]

The following editor-style methods insert item of this type in the position of caret:

- single item: InsertStringTag, InsertStringATag, InsertStringWTag [326];
- multiple items: InsertText, -A, -W [327].

The following editor-style methods change text of the given item:

- SetItemTextEd -A -W [352], SetCurrentItemText -A -W [347]

The following methods return text of item at the position of caret:

- GetCurrentItemText -A -W [313]

The following editor-style methods insert text from files in the position of caret:

- InsertTextFromFile, InsertTextFromFileW [328]

See Also...

See also methods for copying and pasting [48], selecting text [47].

Some of the methods above can insert tabulators [89] as well.

See also: classes for item types [526].

1.2.3 Tabulators

Tabulators

A special item used to align the subsequent text to the next tab stop.

Tab stops are measured from the LeftMargin [145] and defined by the following properties:

- paragraph's LeftIndent [451] (for paragraph with hanging (negative) FirstIndent [451]);
- paragraph's collection of Tabs [455]
- TRVStyle.DefTabWidth [388].

Style [194] of this item type: *rvsTab* (-12)

Properties of this item type are defined by:

- its text style (the same properties as for text items [88])
- corresponding tab stop (item in the paragraph's collection of Tabs [455]: TRVTabInfo [470])

Item of this type can be inserted in document using keyboard (Tab key), or together with text or RTF (Rich Text Format).

Value of SpacesInTab [402] property of the linked TRVStyle [385] component must be 0 (default). If it is positive, several spaces are inserted instead of tabulator.

Editor accepts **Tab** keys only if *rvoWantTabs* is included in EditorOption[291] property.

Tabulators are read from RVF (RichView Format) files and streams regardless of value of SpacesInTab[402] property.

Normally, tabulators are invisible (they can have background color and underline defined by its text style, though). They can be displayed as small arrows if *rvoShowSpecialCharacters* is included in TRichView.Options[146].

Methods of TCustomRichView[128]

Methods that add one text item cannot insert tabulators. They always replace them with several space characters or delete.

The following viewer-style methods can add tabulators (if text contains #09 characters):

- AddTextNL, AddTextNLA, AddTextNLW[171];
- LoadText, LoadTextW[212], LoadTextFromStream, LoadTextFromStreamW[213];
- AddTab[170] – adds tabulator regardless of SpacesInTab mode.

Methods of TCustomRichViewEdit[282]

Methods that add one text item cannot insert tabulators. They always replace them with several space characters or delete.

The following editor-style methods can insert tabulators (if text contains #09 characters):

- InsertText, -A, -W[327], InsertTextFromFile, InsertTextFromFileW[328]
- InsertTab[326] – inserts tabulator regardless of SpacesInTab mode

See also methods for pasting[48].

1.2.4 Pictures

Pictures

Picture item displays a picture stored in TGraphic object.

Delphi has the following standard graphic classes:

- TBitmap (stores bitmap (*.bmp), probably with transparency);
- TMetafile (stores 32-bit (*.emf) or 16-bit (*.wmf) metafile);
- TIcon (stores icon; this class does not work with icons having more than 16 colors properly);
- TJpegImage (D3+; stores Jpegs (*.jpg));
- TGifImage (D2007+, stores Gifs (*.gif));
- TPngImage (D2009+, stores Png (*.png));
- TWicImage (D2010+, by default used for Tiff (*.tif) images, but can be used for other graphic formats).

In addition, you can use third-party graphic classes[671].

Style[194] of this item type: *rvsPicture* (-3)

For hypertext pictures, see *hot-pictures*[93] item type.

Methods of TCustomRichView[128]

The following viewer-style methods add item of this type to the end of the document:

- AddPictureEx[169],
- AddPictureExTag[169]

The following viewer-style method changes the main properties of item of this type:

- SetPictureInfo[234]

The following method returns main properties of the given item of this type:

- GetPictureInfo[201]

Methods of TCustomRichViewEdit 282

The following editor-style method inserts item of this type in the position of caret:

- InsertPicture 322

The following editor-style methods modify main properties of the item of this type:

- SetCurrentPictureInfo 348,
- SetPictureInfoEd 354

The following method returns main properties of the item of this type at the position of caret:

- GetCurrentPictureInfo 315

Properties

Size

This item type has the following integer properties 627 related to size:

- *rvepImageWidth*, *rvepImageHeight* stretch the image.

Layout and appearance

This item type has the following integer properties 627 related to layout and appearance:

- *rvepSpacing* – padding (spacing between the picture and its border; if a background color is specified, this area is colored);
- *rvepColor* – background color;
- *rvepBorderWidth, rvepBorderColor* – width and color of a border;
- *rvepOuterHSpacing, rvepOuterVSpacing* – horizontal and vertical spacing around the border.

Memory usage

By default, TRichView frees a graphic object when this item is destroyed, or when a new graphic object is assigned.

However, if *rvepShared* integer property 627 is nozero, TRichView does not free it.

Vertical Position

This item type has the following properties affecting vertical position:

- VAlign 648;
- extra integer properties 627: *rvepVShift*, *rvepVShiftAbs*.

Resizing

In editor, pictures can be resized with the mouse (except for TIcon). You can forbid mouse resizing by including *rvoNoImageResize* in EditorOptions 291. You can forbid resizing for the specific image by assigning 0 to *rvepResizable* integer property 627.

Resizing with mouse does not modify the image itself, it changes *rvepImageWidth* and *rvepImageHeight*.

OnItemResize 359 event occurs after resizing.

Other properties

This item type has the following specific integer properties 627:

- *rvepTransparent*, *rvepTransparentMode*, *rvepTransparentColor* define transparency in bitmap;
- *rvepAnimationInterval* – interval for bitmap animations;
- *rvepMinHeightOnPage* allows printing this image on multiple pages (cannot be applied to metafiles);
- *rvepNoHTMLImageSize* – if nonzero, <IMG width> and <IMG height> are not saved in HTML for this image.

This item type has the following specific string properties 630:

- *rvespAlt* – text representation of the image, saved in HTML as <IMG alt> attribute;
- *rvespImageFileName* – this property can be saved in HTML file as image location.

Saving and Loading

RichView Format (RVF) [59]

In order to load these items from RVF, you need to register a class of the graphics. RichView does it automatically for the standard item classes. But if you use third-party graphic classes, register them.

For example:

- Delphi: RegisterClass(TMyGifImage);
- C++Builder: RegisterClass(__classid(TMyGifImage));

Classes of the most of standard graphic formats are automatically registered by TRichView, see the topic about RVGraphicHandler [613].

If *rvfoSavePicturesBody* is excluded from RVFOptions [152], graphic data is not saved in RVF. When loading such RVF files, OnRVFPictureNeeded [252] event occurs. You can use this feature to store images in database or files.

Export to HTML [62]

By default, all pictures are saved as JPEGs. If you want to save pictures of some graphic class without conversion, call RVGraphicHandler [662].RegisterHTMLGraphicFormat.

For example:

- Delphi: RVGraphicHandler [662].RegisterHTMLGraphicFormat(TMyGifImage);
- C++Builder: RVGraphicHandler [662].RegisterHTMLGraphicFormat(__classid(TMyGifImage));

The standard PNG and GIF formats may be automatically registered by TRichView, see the topic about RVGraphicHandler [613].

If you want to save pictures in another format, use OnHTMLSaveImage [241] or OnSaveImage2 [260] events.

RTF export

RTFOptions [150] contain options for saving pictures in RTF. Pictures are embedded in RTF, external image files are not created.

Bitmaps are stored as bitmaps. 32-bit metafiles (EMF) are stored as 32-bit or 16-bit (WMF) metafiles.

PNG images are stored as PNG images, if PNG class is specified (see RVGraphicHandler [662] .RegisterPngGraphic).

All other graphic formats are stored as bitmaps, WMF or PNG (except for Jpegs which can be stored as Jpegs).

DocX export

All graphic formats are stored as they are, except for:

- bitmaps (they are saved as PNG, if PNG graphic class is defined, see RVGraphicHandler [662] .RegisterPngGraphic;
- icons (they are saved as PNG in Delphi 2009+ using TPngImage class)

RTF Import

You can set RTFReadProperties [151].IgnorePictures [271] to *True*, and pictures in RTF will not be added in RichView.

RTF can have embedded images and external images (in separate files). In order to load external images, a proper graphic class must be associated with the file extension.

Delphi does it for standard graphic classes automatically, but for some third-party classes you need to do it yourself.

For example:

```
TPicture.RegisterFileFormat('gif','Gif Image',TMyGifImage);
```

Read more information in the topic about third-party graphic classes [671].

See Also...

- Working in 256-color display mode 142;
- Animation 56;
- Smooth image scaling 57.

1.2.5 Hot-Pictures

"Hot-picture" is a picture 90 - hypertext link 35.

Style 194 of this item type: *rvsHotPicture* (-10)

All methods working with pictures 90 work with hot-pictures. The difference is only in methods that insert hot-picture in the document.

Methods of TCustomRichView 128

The following viewer-style methods add item of this type to the end of the document:

- AddHotPicture 163,
- AddHotPictureTag 163

Methods of TCustomRichViewEdit 282

The following editor-style method inserts item of this type at the position of caret:

- InsertHotPicture 319

ConvertToHotPicture 305, ConvertToPicture 305 convert picture to hot-picture and vice versa.

1.2.6 Breaks

Break is a horizontal line (or rectangle) of the specified width and color.

It always occupies a full line.

Style 194 of this item type: *rvsBreak* (-1)

Methods of TCustomRichView 128

The following viewer-style methods add item of this type to the end of the document:

- AddBreakExTag 160 (also AddBreak, AddBreakEx, AddBreakTag)
- The following viewer-style method changes the main properties of item of this type:
- SetBreakInfo 226

The following method returns main properties of the given item of this type:

- GetBreakInfo 183

Methods of TCustomRichViewEdit 282

The following editor-style method inserts item of this type at the position of caret:

- InsertBreak 317

The following editor-style methods modify main properties of the item of this type:

- SetCurrentBreakInfo 342,
- SetBreakInfoEd 339

The following method returns main properties of the item of this type at the position of caret:

- GetCurrentBreakInfo 308

Properties

Main properties of *breaks*: style, width, color.

In addition to the methods listed above, they are accessible as rveipcBreakStyle, rveipcBreakWidth, rveipcBreakColor[658] properties.

Saving and loading

Export to HTML[62]

Breaks are exported as <HR>. Color of *break* is retained only in CSS version (SaveHTMLEx[217], or SaveHTML[216] with *rvsoForceNonTextCSS* in Options parameter)

RTF

Breaks are exported as two paragraphs, one of them with a border at the bottom side. Not imported (i.e. imported back as two paragraphs, one of them with a border at the bottom side)

Export to DocX

Breals are exported as a special line object.

Export to Text

When saving/copying selection, *breaks* are not included in text. When saving to file (SaveText[221]), saved as a sequence of LineWidth '-' characters.

1.2.7 Controls

You can insert any Delphi/C++Builder control in the document.

Style[194] of this item type: *rvsComponent* (-5)

Methods of TCustomRichView[128]

The following viewer-style methods add item of this type to the end of the document:

- AddControlEx[162],
- AddControlExTag[162]

The following viewer-style method changes the main properties of item of this type:

- SetControlInfo[228]

The following method returns main properties of the given item of this type:

- GetControlInfo[187]

See also

- FindControlItemNo[182]

Methods of TCustomRichViewEdit[282]

The following editor-style method inserts item of this type at the position of caret:

- InsertControl[318]

The following editor-style methods modify main properties of the item of this type:

- SetCurrentControlInfo[344],
- SetControlInfoEd[341]

The following method returns main properties of the item of this type at the position of caret:

- GetCurrentControlInfo[309]

Also

- ResizeControl[335],
- ResizeCurrentControl[335],

- AdjustControlPlacement2 [296]

Events:

- OnControlAction [239] – occurs when the control is loaded from RVF [59], deleted, moved to/from undo buffer, saved to RVF.

Properties

Layout and appearance

This item type has the following integer properties [627] related to layout and appearance:

- *rvepSpacing* – padding (spacing between the control and its border; if a background color is specified, this area is colored);
- *rvepColor* – background color;
- *rvepBorderWidth, rvepBorderColor* – width and color of a border;
- *rvepOuterHSpacing, rvepOuterVSpacing* – horizontal and vertical spacing around the border.

Memory usage

By default, TRichView frees a control when this item is destroyed.

However, if *rvepShared* integer property [627] is nozero, TRichView does not free it.

Vertical Position

This item type has the following properties affecting vertical position:

- VAlign [648];
- extra integer properties [627]: *rvepVShift*, *rvepVShiftAbs*.

Resizing

You can allow resizing this control in editor by assigning non-zero value to its *rvepResizable* property [627].

You can forbid mouse resizing of all images and controls in the editor by including *rvoNoImageResize* in EditorOptions [291].

Resizing with mouse changes Width and Height properties of the control.

Resizing is activated if this control is selected. You can select this control (for example, in its OnClick event) by calling RichView.SelectControl [223].

OnItemResize [359] event occurs after resizing.

Other properties

This item type has the following specific integer properties [627]:

- *rvepVisible* allows to hide the control (to set its Visible property to *False*).

Note about Tags of controls

RichView uses tags of inserted controls (TComponent.Tag) for its own needs. Please do not use and do not modify tags of controls.

(this note is about Tag property of component, not about tags of items [34]; you can use item's tag as you wish)

Saving and Loading

RVF [59]

In order to load these items from RVF, you need to register a class of the control.

For example:

- Delphi: RegisterClass(TButton);
- C++Builder: RegisterClass(__classid(TButton));

If *rvfoSaveControlsBody* is excluded from RVFOptions [152], control itself is not saved in RVF. When loading such RVF files, OnRVFControlNeeded [251] event occurs.

Export to text, RTF, and HTML 62

By default, controls are not exported. Use OnSaveComponentToFile 256 event to save controls.

Printing

RichView creates a temporal bitmap, draws the control in it, then prints this bitmap.

It's not possible to create an universal procedure for drawing all possible controls. If you are not satisfied with the results, override the default printing with OnPrintComponent 519 event.

Special Width Mode

Width of controls can be set in percent:

```
var Item: TRVControlItemInfo;
  ...
Item := RichView.GetItem(ItemNo) as TRVControlItemInfo;
Item.PercentWidth := 50; // 50% of document width (minus paragraph indents)
```

The effect will be after reformatting 182.

See also...

BeginOleDrag 174

1.2.8 Bullets

"Bullet" is an image from TImageList. *Bullets* are useful if you want to create document where the same (small) image is used many times.

For example, they can be used for smiles in chats.

This item type has no relation to paragraph bullets and numbering 99.

Style 194 of this item type: *rvsBullet* (-6)

For hypertext *bullets*, see *hotspot* 97 item type.

Methods of TCustomRichView 128

The following viewer-style methods add item of this type to the end of the document:

- AddBulletEx, AddBulletExTag 160

The following viewer-style method changes the main properties of item of this type:

- SetBulletInfo 227

The following method returns main properties of the given item of this type:

- GetBulletInfo 184

Methods of TCustomRichViewEdit 282

The following editor-style method inserts item of this type at the position of caret:

- InsertBullet 317

The following editor-style methods modify main properties of the item of this type:

- SetCurrentBulletInfo 342,
- SetBulletInfoEd 339

The following method returns main properties of the item of this type at the position of caret:

GetCurrentBulletInfo 308

Properties

Main properties

ImageList is specified when this item is added. It cannot be changed.

Image index can be changed by the methods listed above. It is also accessible as rveipcImageIndex 658 property.

Layout and appearance

This item type has the following integer properties 627 related to layout and appearance:

- *rvepSpacing* – padding (spacing between the picture and its border; if a background color is specified, this area is colored);
- *rvepColor* – background color;
- *rvepBorderWidth, rvepBorderColor* – width and color of a border;
- *rvepOuterHSpacing, rvepOuterVSpacing* – horizontal and vertical spacing around the border.

Vertical Position

This item type has the following properties affecting vertical position:

- VAlign 648;
- extra integer properties 627: *rvepVShift*, *rvepVShiftAbs*.

Resizing

Bullets cannot be resized.

Other properties

This item type has the following specific string properties 630:

- *rvespAlt* – text representation of the image, saved in HTML as <IMG alt> attribute;

Memory Usage

RichView does not own this image-list, and it does not destroy it when clearing document.

Saving and Loading

RVF 59

ImageLists themselves are not saved in RVF. RichView stores TImageList.Tag. When loading, OnRVFImageListNeeded 252 occurs allowing you to provide image-lists for *bullets* and *hotspots*.

Export to RTF and HTML 62

Bullets are exported as pictures. If you want to write special code for exporting bullets in HTML, use OnHTMLSaveImage 241 event.

1.2.9 Hotspots

"Hotspot" is a *bullet* 96 (image from TImageList) with hypertext link 35.

Hotspots can change their image under the mouse pointer.

Style 194 of this item type: *rvsHotspot* (-4)

Methods of TCustomRichView 128

The following viewer-style methods add item of this type to the end of the document:

- AddHotspotEx 164,
- AddHotspotExTag 164

The following viewer-style method changes the main properties of item of this type:

- SetHotspotInfo 229

The following method returns main properties of the given item of this type:

- GetHotspotInfo 189,
- GetBulletInfo 184

Methods of TCustomRichViewEdit 282

The following editor-style method inserts item of this type at the position of caret:

- InsertHotspot 319

The following editor-style methods modify main properties of the item of this type:

- SetCurrentHotspotInfo 345,
- SetHotspotInfoEd 349

The following method returns main properties of the item of this type at the position of caret:

- GetCurrentHotspotInfo 310,
- GetCurrentBulletInfo 308

Properties

Hotspots have all properties of *bullets*.

Image index below the mouse pointer can be changed by the methods listed above. It is also accessible as rveipcHotImageIndex 658 property.

Memory Usage

RichView does not own this image-list, and it will not destroy it when clearing document.

1.2.10 Tables

Tables arrange content in rows and columns. Each table cell is a subdocument containing its own items. Tables can be nested.

More details:

- Overview of tables 111,
- **TRVTableItemInfo** 528 (see for detailed information).

Style 194 of this item type: *rvsTable* (-60)

There is an important differences between properties of tables and properties of any other items. To change values of properties of any other item as an editing operation, you need to use special methods. But direct assignment to properties of TRVTableItemInfo is written to an undo buffer, if this table is already inserted in TCustomRichViewEdit 282.

Extra Properties

This item type has the following specific string properties 630:

- *rvespImageFileName* – value of this property can be saved in HTML file as table background image location; also accessible as table.BackgroundImageFileName 531.

Methods of TCustomRichView 128

The following viewer-style methods add item of this type to the end of the document:

- AddItem 166 (create TRVTableItemInfo 528 object, add it using AddItem 166)

The following method returns TRVTableItemInfo 528 object

- GetItem 190

Methods of TCustomRichViewEdit 282

The following editor-style method inserts item of this type in the position of caret:

- InsertItem 320 (create TRVTableItemInfo 528.object, insert it using InsertItem 320)

The following methods return TRVTableItemInfo 528 object at the position of caret:

- GetCurrentItem 311,
- GetCurrentItemEx 311.

Resizing

Users can resize table rows and columns, if *rvtoRowSizing* and/or *rvtoColSizing* are included in table.Options 540 .

Resizing changes BestWidth 569 and BestHeight 568 properties of table cells.

OnItemResize 359 event occurs after resizing.

Saving and Loading

Export to HTML 62

The TRichView's model of tables is very similar to the HTML model of tables, so tables are saved to HTML as they are.

The most of table visual properties can be saved only with use of CSS (use SaveHTMLEx 217 method, or SaveHTML 216 with *rvsoForceNonTextCSS* in Options parameter).

RTF Import

You can set RTFReadProperties 151.IgnoreTables 272 to *True*, and tables in RTF will not be added in RichView (contents of cells will be added in the main document).

RTF tables have different structure, TRichView does a smart work for converting them.

Text export

See the following properties of TRVTableItemInfo 528:

- TextColSeparator 543;
- TextRowSeparator 544.

1.2.11 List Markers

This is a very special item type representing paragraph bullet or number.

Style 194 of this item type: *rvsListMarker* (-11)

See also: numbered sequences 102.

Properties of List Markers

TRVStyle 385 component has a collection of list styles (ListStyles 396). Each list style 459 contains a collection of list levels (Levels 459 property). List level 472 defines the most of list properties.

List markers is a special kind of items. They are inserted with special functions. They are always inserted at the beginning of paragraph.

They have the following properties:

- ListNo – index of list style in the collection of list styles 396;
- LevelNo – list level;
- StartFrom – starting value for list counter, if UseStartFrom=True;
- UseStartFrom – if True, list counter value for this marker is defined by StartFrom. If False, numbering is continued.

List Markers Operations

Methods of TCustomRichView:

- SetListMarkerInfo 233,
- GetListMarkerInfo 199,
- RefreshListMarkers 214.

Methods of TCustomRichViewEdit:

- ApplyListStyle 297,
- RemoveLists 334,
- ChangeListLevels 304.

Saving and Loading

RVF 59

For list levels with ImageLists: ImageLists themselves are not saved in RVF. RichView stores TImageList.Tag. When loading, OnRVFImageListNeeded 252 occurs allowing you to provide an image-list.

HTM Export, rvsoMarkersAsText is not in Options of SaveHTML 216 or SaveHTMLEx 217

Lists are saved using <ul> and <ol> HTML tags.

Export using CSS (SaveHTMLEx 217):

CSS of lists are inserted directly in HTML.

- rvlstBullet, rvlstUnicodeBullet 476 are exported as bullets with default markers (disc, circle or square). FormatString 474 and Font 473 are ignored;
- rvlstDecimal,rvlstLowerAlpha,rvlstUpperAlpha,rvlstLowerRoman,rvlstUpperRoman 476 are exported as lists with corresponding numbering. FormatString 474 and Font 473 are ignored;
- rvlstPicture,rvlstImageList,rvlstImageListCounter 476 are exported as bullets with picture. It's possible to change default image saving by OnHTMLSaveImage 241.

Indents are saved, but not very accurately.

Export without CSS (SaveHTML 216)

- rvlstBullet, rvlstUnicodeBullet,rvlstPicture,rvlstImageList 476 are exported as bullets with default markers (disc, circle or square). FormatString 474, Picture 479, ImageList 475 and Font 473 are ignored;
- rvlstDecimal,rvlstLowerAlpha,rvlstUpperAlpha,rvlstLowerRoman,rvlstUpperRoman 476 are exported as lists with corresponding numbering. FormatString 474 and Font 473 are ignored;
- rvlstImageListCounter 476 is saved as a decimal numbering.

Indents are not saved.

HTML Export, rvsoMarkersAsText is in Options

Markers are saved without using special HTML keywords for lists (like <ol>, <ul>, <li>).

Font 473 and format strings 474 settings are respected.

Saving with CSS (SaveHTMLEx 217): LeftIndent 476 and MarkerIndent 478 are retained. FirstIndent 473 are retained for non-hanging markers only (MarkerIndent>=LeftIndent).

Saving without CSS (SaveHTML 216): all indents are ignored.

Generally, HTML files saved with this option look closer to the original.

Demo Projects

- **Demos\Delphi\Assorted\ListStyles**
- **Demos\CBuilder\Assorted\ListStyles**

1.2.12 Labels

"Label" is a non-text item looking like text [88]. This text cannot be wrapped (always displayed on one line).

For Delphi/C++Builder 2009 or newer, this text is Unicode [64]. For older versions, it is ANSI.

Class for this item type is **TRVLabelItemInfo** [579] (see for detailed information).

Style [194] of this item type: *rvsLabel* (-200)

The following items are inherited from labels:

- numbered sequences [102]
- footnotes [104]
- endnotes [103]
- sidenotes [105]
- text boxes [107]
- references to notes [108]

Related Properties of TRVStyle [385]

- FieldHighlightColor [390] – color for highlighting label items.
- FieldHighlightType [390] specifies when to highlight label items.

Methods of TCustomRichView [128]

The following viewer-style method adds item of this type to the end of the document:

- AddItem [166] (create TRVLabelItemInfo [579] object, add it using AddItem [166])

The following method returns TRVLabelItemInfo [579] object

- GetItem [190]

Methods of TCustomRichViewEdit [282]

The following editor-style method inserts item of this type in the position of caret:

- InsertItem [320] (create TRVLabelItemInfo [579] object, insert it using InsertItem [320])

The following method returns TRVLabelItemInfo [579] object at the position of caret:

- GetCurrentItem [311].

The following methods change properties as editing operations:

- SetCurrentItemExtraIntPropertyEx [345];
- SetItemExtraIntPropertyExEd [350];
- SetCurrentItemExtraStrPropertyEx [346];
- SetItemExtraStrPropertyExEd [351].

TextStyleNo [581] property can be modified by ApplyTextStyle [300] and ApplyStyleConversion [299] methods.

Properties

Main properties

- Text [581] – text to display;
- TextStyleNo [581] defines font and colors. This is an index in TextStyles [404] collection of the linked TRVStyle [385] component;
- ProtectTextStyleNo [581] – if *True*, editing operations cannot change TextStyleNo [581];
- MinWidth [580] – minimal width for the label item;
- Alignment [580] – horizontal alignment of text (if the item width is increased by MinWidth [580]);
- Cursor [580] – mouse cursor above this item.
- RemoveInternalLeading [581] – if *True*, the font internal leading is not included in the item height.

These properties are also accessible as rvespcText [661], rveipcProtectTextStyleNo, rveipcMinWidth, rveipcAlignment, rveipcCursor, rveipcRemoveInternalLeading [658] properties. Accessing them in this way allows changing them in editing operations.

Layout and appearance

This item type has the following integer properties [627] related to layout and appearance:

- *rvepSpacing* – padding (spacing between the text and its border; if a background color is specified, this area is colored);
- *rvepColor* – background color;
- *rvepBorderWidth, rvepBorderColor* – width and color of a border;
- *rvepOuterHSpacing, rvepOuterVSpacing* – horizontal and vertical spacing around the border.

Vertical Position

This item type has the following properties affecting vertical position:

- VAlign [648];
- extra integer properties [627]: *rvepVShift*, *rvepVShiftAbs*.

1.2.13 Numbered Sequences

Numbered sequence is a non-text item looking like text [88] and displaying a number. For example, this item type can be used for numbering pictures, tables or formulas. This text cannot be wrapped (always displayed on one line).

Class for this item type is **TRVSeqItemInfo** [582] (see for detailed information).

Style [194] of this item type: *rvsSequence* (-202)

The following items are inherited from numbered sequences:

- footnotes [104]
- endnotes [103]
- sidenotes [105]
- text boxes [107]

Methods of TCustomRichView [128]

The following viewer-style method adds item of this type to the end of the document:

- AddItem [166] (create TRVSeqItemInfo [582] object, add it using AddItem [166])

The following method returns TRVSeqItemInfo [582] object

- GetItem [190]

Methods of TCustomRichViewEdit [282]

The following editor-style method inserts item of this type in the position of caret:

- InsertItem [320] (create TRVSeqItemInfo [582] object, insert it using InsertItem [320])

The following method returns TRVSeqItemInfo [582] object at the position of caret:

- GetCurrentItem [311].

The following methods change properties of items as editing operations:

- SetCurrentItemExtraIntPropertyEx [345];
- SetItemExtraIntPropertyExEd [350];
- SetCurrentItemExtraStrPropertyEx [346];
- SetItemExtraStrPropertyExEd [351].

Properties

Numbered sequences have all properties of label items [101], but Text [581] is calculated automatically.

Additionally, it has properties:

- NumberType [584] – numbering type;
- SeqName [585] – name of a sequence (this item continues numbering from the previous item having the same name of a sequence);

- Reset 584 – if *True*, a numbering is not continued but reset from StartFrom 585.
- FormatString 584 – optional format string; if defined, a text to display is formatted according to this string.

These properties are also accessible as rvespcSeqName, rvespcSeqFormatString 661, rveipcSeqStartFrom, rveipcSeqReset, rveipcSeqNumberType 658 properties. Accessing them in this way allows changing them in editing operations.

Saving and Loading

Export to HTML 62 and text

Numbered sequences are exported as a plain text.

RTF

Numbered sequences can be exported in RTF (all main properties are saved except for FormatString 584).

You can set RTFReadProperties 151.IgnoreSequences 272 to *True*, and numbered sequences will not be imported (imported as a plain text) from RTF.

1.2.14 Endnotes

An endnote looks like a number (numbering type is specified in TRVStyle.EndnoteNumbering 389). If FormatString 584 is defined, a number is displayed formatted according to this string (for example, if FormatString is a single character, the endnote displays a single character). Endnote contains a subdocument. This subdocument is not displayed in TCustomRichView where the endnote is inserted, but it can be printed. Subdocuments for all endnotes are printed after the main document.

ScaleRichView note: TSRichViewEdit displays and allows editing endnotes. TSRichViewEdit cannot print a text for an endnote on several pages.

Class for this item type is **TRVEndnoteItemInfo** 587 (see for detailed information).

Style 194 of this item type: *rvsEndnote* (-204)

Methods of TCustomRichView 128

The following viewer-style method adds item of this type to the end of the document:

- AddItem 166 (create TRVEndnoteItemInfo 587 object, add it using AddItem 166)

The following method returns TRVEndnoteItemInfo 587 object

- GetItem 190

Methods of TCustomRichViewEdit 282

The following editor-style method inserts item of this type in the position of caret:

- InsertItem 320 (create TRVEndnoteItemInfo 587 object, insert it using InsertItem 320)

The following method returns TRVEndnoteItemInfo 587 object at the position of caret:

- GetCurrentItem 311.

The following methods change properties of items as editing operations:

- SetCurrentItemExtraIntPropertyEx 345;
- SetItemExtraIntPropertyExEd 350;
- SetCurrentItemExtraStrPropertyEx 346;
- SetItemExtraStrPropertyExEd 351.

Properties

Endnotes have all properties of numbered sequences 102.

Differences:

- NumberType 584 is read-only; it always returns TRVStyle.EndnoteNumbering 389;

- SeqName 585 = '@endnote@' and must not be changed.

Saving and Loading

Export to HTML 62

Subdocuments for all endnotes are saved after the main document. Before each subdocument, is inserted, where *N* is an index of this endnote (in the list of all numbered sequences 102 and notes). Endnotes are saved as hyperlinks to this anchor.

RTF

Endnotes can be exported in RTF. If FormatString 584 contains a single character, it can be exported to RTF. Reset 584 and StartFrom 585 properties cannot be exported.

You can set RTFReadProperties 151.IgnoreNotes 271 to *True*, and endnotes will not be imported from RTF.

DocX

Endnotes can be exported in DocX. If FormatString 584 contains a single character, it can be exported to DocX.

Text

When copying or saving a selected text, endnotes are not included in text. When saving the whole text, they are saved at the end of the text:

```
This is a text [i] to show endnotes [ii].
-----------------
i sequence of characters
ii note of text placed at the end of the document
```

1.2.15 Footnotes

A footnote looks like a number (numbering type is specified in TRVStyle.FootnoteNumbering 389). If FormatString 584 is defined, a number is displayed formatted according to this string (for example, if FormatString is a single character, the footnote displays a single character).

Footnote contains a subdocument. This subdocument is not displayed in TCustomRichView where the footnote is inserted, but it can be printed. Subdocuments for all footnotes are printed at the bottom of page where this footnote occurs. TRichView cannot print text for a footnote on several pages.

ScaleRichView note: TSRichViewEdit displays and allows editing footnotes.

Class for this item type is **TRVFootnoteItemInfo** 589 (see for detailed information).

Style 194 of this item type: *rvsFootnote* (-203)

Methods of TCustomRichView 128

The following viewer-style method adds item of this type to the end of the document:

- AddItem 166 (create TRVFootnoteItemInfo 589 object, add it using AddItem 166)

The following method returns TRVFootnoteItemInfo 589 object

- GetItem 190

Methods of TCustomRichViewEdit 282

The following editor-style method inserts item of this type in the position of caret:

- InsertItem 320 (create TRVFootnoteItemInfo 589 object, insert it using InsertItem 320)

The following method returns TRVFootnoteItemInfo 589 object at the position of caret:

- GetCurrentItem 311.

The following methods change properties of items as editing operations:

- SetCurrentItemExtraIntPropertyEx 345;

- SetItemExtraIntPropertyExEd 350;
- SetCurrentItemExtraStrPropertyEx 346;
- SetItemExtraStrPropertyExEd 351.

Properties

Footnotes have all properties of numbered sequences 102.

Differences:

- NumberType 584 is read-only; it always returns TRVStyle.FootnoteNumbering 389;
- SeqName 585 = '@footnote@' and must not be changed.

Saving and Loading

Export to HTML 62

Subdocuments for all footnotes are saved after the main document. Before each subdocument, is inserted, where *N* is an index of this footnote (in the list of all numbered sequences 102 and notes). Footnotes are saved as hyperlinks to this anchor.

RTF

Footnotes can be exported in RTF. If FormatString 584 contains a single character, it can be exported to RTF.

You can set RTFReadProperties 151.IgnoreNotes 271 to *True*, and footnotes will not be imported from RTF.

DocX

Footnotes can be exported in DocX. If FormatString 584 contains a single character, it can be exported to DocX.

Text

When copying or saving a selected text, footnotes are not included in text. When saving the whole text, they are saved inside [], for example:

```
This is a text [1 sequence of characters] to show footnotes [2 note of text placed at the
```

1.2.16 Sidenotes

A sidenote looks like a number (numbering type is specified in TRVStyle.SidenoteNumbering 389). If FormatString 584 is defined, a number is displayed formatted according to this string.

Sidenote contains a subdocument. This subdocument is not displayed in TCustomRichView where the sidenote is inserted, but it can be printed inside a floating box.

ScaleRichView note: TSRichViewEdit displays and allows editing sidenotes.

Class for this item type is **TRVSidenoteItemInfo** 591 (see for detailed information).

Style 194 of this item type: *rvsSidenote* (-206)

The following items are inherited from numbered sidenotes:

- text boxes 107

Methods of TCustomRichView 128

The following viewer-style method adds item of this type to the end of the document:

- AddItem 166 (create TRVSidenoteItemInfo 591 object, add it using AddItem 166)

The following method returns TRVSidenoteItemInfo 591 object

- GetItem 190

Methods of TCustomRichViewEdit 282

The following editor-style method inserts item of this type in the position of caret:

- InsertItem 320 (create TRVSidenoteItemInfo 591 object, insert it using InsertItem 320)

The following method returns TRVSidenoteItemInfo[591] object at the position of caret:

- GetCurrentItem[311].

The following methods change properties of items as editing operations:

- SetCurrentItemExtraIntPropertyEx[345];
- SetItemExtraIntPropertyExEd[350];
- SetCurrentItemExtraStrPropertyEx[346];
- SetItemExtraStrPropertyExEd[351].

Properties

Sidenotes have all properties of numbered sequences[102].

Differences:

- NumberType[584] is read-only; it always returns TRVStyle.SidenoteNumbering[389];
- SeqName[585] = '@sidenote@' and must not be changed.

Additionally, it has the following properties:

- BoxPosition[596] – properties relative to the position of a text box;
- BoxProperties[593] – size, border, background, content alignment of a text box.

These properties are also accessible as rveipcFloatHorizontalAnchor, rveipcFloatHorizontalPositionKind, rveipcFloatHorizontalOffset, rveipcFloatHorizontalAlign, rveipcFloatVerticalAnchor, rveipcFloatVerticalPositionKind, rveipcFloatVerticalOffset, rveipcFloatVerticalAlign, rveipcFloatRelativeToCell, rveipcFloatPositionInText, rveipcBoxWidth, rveipcBoxHeight, rveipcBoxWidthType, rveipcBoxHeightType, rveipcBoxBorderColor, rveipcBoxBorderWidth, rveipcBoxBorderInternalWidth, rveipcBoxBorderStyle, rveipcBoxBorderVisibleBorders_Left, rveipcBoxBorderVisibleBorders_Top, rveipcBoxBorderVisibleBorders_Right, rveipcBoxBorderVisibleBorders_Bottom, rveipcBoxBorderBorderOffsets_Left, rveipcBoxBorderBorderOffsets_Top, rveipcBoxBorderBorderOffsets_Right, rveipcBoxBorderBorderOffsets_Bottom, rveipcBoxColor, rveipcBoxPadding_Left, rveipcBoxPadding_Top, rveipcBoxPadding_Right, rveipcBoxPadding_Bottom, rveipcBoxVAlign properties[658]. Accessing them in this way allows changing them in editing operations.

Saving and Loading

Export to HTML[62]

Subdocuments for all sidenotes are saved after the main document. Before each subdocument, is inserted, where *N* is an index of this sidenote (in the list of all numbered sequences[102] and notes). Sidenotes are saved as hyperlinks to this anchor.

RTF

Sidenotes can be exported in RTF. They are imported as NoteText[587] followed by a text box containing Document[586]. A bookmark named '_sidenoteN' (where *N* is an index of this sidenote) is added around NoteText, to allow saving note references[108] as references to this bookmark.

Some combinations of box positioning properties are not supported by Microsoft Word*, so TRichView converts them:

- MS Word does not support a vertical alignment relative to a paragraph, so it saved as an alignment relative to a line;
- MS Word does not support a relative (percent) positioning to a line or paragraph, so it is saved as an absolute positioning (with zero offset);

Other differences*:

- when a line is chosen as an anchor object, MS Word aligns a text box not relative to a whole line but relative to a line top, so a center and a bottom alignment relative to a line are different in TRichView and in MS Word;
- MS Word does not implement all options of vertical alignment relative to a table cell, so it works differently in TRichView and MS Word, see BoxPosition[596].VerticalAnchor[599];
- MS Word does not support different padding for border and background, so BoxProperties[593].Border[594].BorderOffsets[467] are not saved to RTF.

(* this information is based on tests of MS Word 2013)

Floating boxes cannot be imported from RTF (yet).

DocX

Sidenotes can be exported in DocX, both in the old (VML) and the new (DrawingML) formats.

See the comments about RTF.

Text

When copying or saving a selected text, sidenotes are not included in text. When saving the whole text, they are saved inside [], for example:

```
This is a text [a sequence of characters] to show sidenotes [b note of text placed in a f
```

1.2.17 Text Boxes

A text box item is a simplified version of a sidenote [105]. There is no visible text at the place of insertion of this item.

If rvoShowSpecialCharacters is included in RichView.Options [146], a special marks are displayed in places of insertion of text box items.

Sidenote contains a subdocument. This subdocument is not displayed in TCustomRichView where the text box item is inserted, but it can be printed inside a floating box.

ScaleRichView note: TSRichViewEdit displays and allows editing text boxes.

Class for this item type is **TRVTextBoxItemInfo** [601] (see for detailed information).

Style [194] of this item type: *rvsTextBox* (-207)

Methods of TCustomRichView [128]

The following viewer-style method adds item of this type to the end of the document:

- AddItem [166] (create TRVSidenoteItemInfo [591] object, add it using AddItem [166])

The following method returns TRVSidenoteItemInfo [591] object

- GetItem [190]

Methods of TCustomRichViewEdit [282]

The following editor-style method inserts item of this type in the position of caret:

- InsertItem [320] (create TRVSidenoteItemInfo [591] object, insert it using InsertItem [320])

The following method returns TRVSidenoteItemInfo [591] object at the position of caret:

- GetCurrentItem [311].

The following methods change properties of items as editing operations:

- SetCurrentItemExtraIntPropertyEx [345];
- SetItemExtraIntPropertyExEd [350];
- SetCurrentItemExtraStrPropertyEx [346];
- SetItemExtraStrPropertyExEd [351].

Properties

Text box items have all properties of sidenotes [105], but almost all of them are ignored.

Only floating box-related properties are used: BoxPosition [596] and BoxProperties [593].

Saving and Loading

Export to HTML [62]

Text boxes are not exported to HTML.

RTF and DocX

Text boxes are exported to RTF and DocX.

See the comments about export of sidenotes[105] for compatibility with Microsoft Word.

Text

When copying or saving a selected text, text boxes are not included in text. When saving the whole text, they are saved inside [], for example:

```
This is a text[sequence of characters] to show text boxes[text placed in a floating box].
```

1.2.18 References to Notes

When inserted in footnote[104]'s/endnote[103]'s/sidenote[105]'s Document[586], the note reference displays the same text as the parent note.

When inserted in TCustomRichView, it displays NoteText[146].

Class for this item type is **TRVNoteReferenceItemInfo**[602] (see for detailed information).

Style[194] of this item type: *rvsNoteReference* (-205)

Methods of TCustomRichView[128]

The following viewer-style method adds item of this type to the end of the document:

- AddItem[166] (create TRVNoteReferenceItemInfo[602] object, add it using AddItem[166])

The following method returns TRVNoteReferenceItemInfo[602] object

- GetItem[190]

Methods of TCustomRichViewEdit[282]

The following editor-style method inserts item of this type in the position of caret:

- InsertItem[320] (create TRVNoteReferenceItemInfo[602] object, insert it using InsertItem[320])

The following method returns TRVNoteReferenceItemInfo[602] object at the position of caret:

- GetCurrentItem[311].

The following methods change properties of items as editing operations:

- SetCurrentItemExtraIntPropertyEx[345];
- SetItemExtraIntPropertyExEd[350];
- SetCurrentItemExtraStrPropertyEx[346];
- SetItemExtraStrPropertyExEd[351].

Saving and Loading

Export to HTML[62]

Note references are saved as a plain text. In future, they may be saved as hyperlinks to the parent footnote/endnote.

RTF

Note references can be exported in RTF.

You can set RTFReadProperties[151].IgnoreNotes[271] to *True*, and note references will not be imported from RTF.

Export to DocX

Note references can be exported in DocX

1.2.19 Page Numbers

When inserted in TRichView[128] or TRichViewEdit[282], this item is displayed as '{p}'.

When printing with TRVPrint[479], this item displays a page number where this item is located. Page numbers are counted from TRVPrint.PageNoFromNumber[516].

When drawing with TRVReportHelper[503], the default behavior is like in TRVPrint, but you can provide a value for the page number in OnGetPageNumber[508] event.

ScaleRichView: When inserted in TSRichViewEdit, this item displays a page number where this item is located. Page numbers are counted from TSRichViewEdit.PageProperty.PageNoFromNumber.

You can insert this item in the main document, a note, a text box, a header or a footer.

Class for this item type is **TRVPageNumberItemInfo**[605] (see for detailed information).

Style[194] of this item type: *rvsPageNumber* (-208)

Methods of TCustomRichView[128]

The following viewer-style method adds item of this type to the end of the document:

- AddItem[166] (create TRVPageNumberItemInfo[605] object, add it using AddItem[166])

The following method returns TRVPageNumberItemInfo[605] object

- GetItem[190]

Methods of TCustomRichViewEdit[282]

The following editor-style method inserts item of this type in the position of caret:

- InsertItem[320] (create TRVPageNumberItemInfo[605] object, insert it using InsertItem[320])

The following method returns TRVPageNumberItemInfo[605] object at the position of caret:

- GetCurrentItem[311].

The following methods change properties of items as editing operations:

- SetCurrentItemExtraIntPropertyEx[345];
- SetItemExtraIntPropertyExEd[350];
- SetCurrentItemExtraStrPropertyEx[346];
- SetItemExtraStrPropertyExEd[351].

Properties

Fields have all properties of label items[101].

By default, Text[581]='{p}'. You can change this value to display another code in TRichView and TRichViewEdit.

Additionally, it has property:

- NumberType[605] – numbering type;

This property is also accessible as rveipcPageNumberType[658] property. Accessing it in this way allows changing it in editing operations.

Saving and Loading

Export to HTML[62]

Fields are saved as a plain text.

RTF

Fields can saved to RTF and loaded from RTF. You can disallow loading from RTF by assinging TRichView.RTFReadProperties[151].IgnoreFields[271]=*True*.

Export to DocX

Fields can be exported in DocX

1.2.20 Page Counts

When inserted in TRichView[128] or TRichViewEdit[282], this item is displayed as '{P}'.

When printing with TRVPrint[479], this item displays the count of pages.

ScaleRichView: When inserted in TSRichViewEdit, this item displays the count of pages.

Class for this item type is **TRVPageNumberItemInfo**[605] (see for detailed information).

Style[194] of this item type: *rvsPageCount* (-209)

Methods of TCustomRichView[128]

The following viewer-style method adds item of this type to the end of the document:

- AddItem[166] (create TRVPageCountItemInfo[607] object, add it using AddItem[166])

The following method returns TRVPageCountItemInfo[607] object

- GetItem[190]

Methods of TCustomRichViewEdit[282]

The following editor-style method inserts item of this type in the position of caret:

- InsertItem[320] (create TRVPageCountItemInfo[607] object, insert it using InsertItem[320])

The following method returns TRVPageCountItemInfo[607] object at the position of caret:

- GetCurrentItem[311].

The following methods change properties of items as editing operations:

- SetCurrentItemExtraIntPropertyEx[345];
- SetItemExtraIntPropertyExEd[350];
- SetCurrentItemExtraStrPropertyEx[346];
- SetItemExtraStrPropertyExEd[351].

Properties

Fields have all properties of label items[101].

By default, Text[581]='{P}'. You can change this value to display another code in TRichView and TRichViewEdit.

Additionally, it has property:

- NumberType[605] – numbering type;

This property is also accessible as rveipcPageNumberType[658] property. Accessing it in this way allows changing it in editing operations.

Saving and Loading

Export to HTML[62]

Fields are saved as a plain text.

RTF

Fields can saved to RTF and loaded from RTF. You can disallow loading from RTF by assinging TRichView.RTFReadProperties[151].IgnoreFields[271]=*True*.

Export to DocX

Fields can be exported in DocX

Note: If RTF or DocX file is opened in Microsoft Word, this field is displayed as "?" until fields are updated.

1.2.21 Custom Item Types

You can create your own item types.

Overview of custom item types is beyond the scope of this help file.

See examples in **Demos\Addins**

1.3 Overview of Tables

I want to thank EQL International (the UK's leading provider of e-learning and assessment systems) and personally Dr. David G Wild.

Without them I could spend years implementing tables.

Overview of Tables

- Tables in RichView 111
- Addressing Items in Table Cells (important information) 112
- Table Cells 113
- Colors and Layout of Tables 114
- Selection in Table 116
- Cell Editing 117
- Cell Merging 118
- Table Operations 119
- Undo/redo in Tables 121
- Export of Tables 123
- Table Resizing 124

Examples

- Converting table to text 125
- Dividing table 126

Main Classes

- TRVTableItemInfo 528 – table
- TRVTableCellData 567 – table cell

1.3.1 Tables in RichView

Table is a special kind of RichView items 29, like images, controls, breaks, etc. But tables are much more complicated, because they contain RichView document in each cell. Moreover, tables can contain other tables inside, so RichView documents can be nested on any depth.

How to Create and Insert a Table

Creating table

```
uses RVTable;
...
var table:TRVTableItemInfo 528;
...
table := TRVTableItemInfo.CreateEx 549(RowCount,ColumnCount,
  RVData);
```

where RVData is a richview document where you wish to insert table. Usually this is richview.RVData 151 or richviewedit.RVData.

How to append table to RichView[128]

```
table.ParaNo := <Paragraph Style Index>
richview.AddItem[166](<Table name>, table)
```

where <table name> is any string associated with table, <paragraph style index> is the index in the collection of paragraph styles[398]. This paragraph style is used for the table itself, not for each cells.

This method does not perform reformatting and requires explicit call of Format[182] before displaying the document.

How to insert table in RichViewEdit[282]

```
richviewedit.InsertItem[320](<Table name>, table)
```

This method inserts table in the position of caret, even if caret is in a cell of another table.

This method can return False, if document is read-only or insertion is impossible because of text protection. In this case table was freed by InsertItem method.

Table Placement

Table always occupies the whole line, no items are possible to the left or to the right of it (if you need this effect, use nested tables), including list markers[99].

The table's paragraph style defines its position: Alignment[449], LeftIndent[451], RightIndent[454] (FirstIndent[451] is ignored). The paragraph style can define border[451] around the table and background[450]. All these properties are applied to the table itself, not to its cells.

1.3.2 Addressing Items in Table Cells

Main Methods

RichViewEdit.[52] **Insert***[52] methods[52] insert item(s) in the position of caret, even if the caret is in cell of other table.

RichViewEdit.**GetCurrent***Info** and **SetCurrent***Info** methods differ from their analogs with names without "Current"

(for example GetCurrentPictureInfo[315] and GetPictureInfo[201])

Get/SetCurrent*Info** methods work with the current item (item at the position of caret) either in parent editor or in cell inplace-editor, depending on where the caret is located.

Get/Set-without-Current methods cannot work with cells (because item in cell cannot be addressed using only ItemNo parameter).

If you used TRichView version 1.3 and you want to work with tables, you should check your code and replace such methods with their "Current" alternatives, or to call them for TopLevelEditor[295].

All methods (and properties) of editor working with current or selected item(s) can work with cell inplace editors

(for example, ApplyTextStyle[300], SetCurrentTag[349], CurTextStyleNo[290], etc.). Programmer does not need to know if the current item or selection is in root editor or in inplace editor.

Methods with ItemNo Parameter

Methods working with the Item-th item cannot work with cells (because item in cell cannot be addressed using only ItemNo parameter). They must be called for the document containing this item (usually, for editing operations, it's TopLevelEditor[295]).

There are some methods which do not have ItemNo parameter:

- ResizeCurrentControl[335] (alternative to ResizeControl[335])
- AdjustControlPlacement2[296](Control) (alternative to AdjustControlPlacement[296](ItemNo))
- SetCurrentItemText[347](text); (alternative to SetItemTextEd[352](ItemNo, text));

- GetCurrentItemText 313 (alternative to GetItemText 195);

Events with ItemNo Parameter

For events without RVData 608 parameter: when events occur inside cell inplace editor, ItemNo is an index of table inside the root editor (for example OnRVMouseDown 253)

For events having both RVData and ItemNo parameter: ItemNo is an index of item in this RVData

Selection

SelectionExists 224 returns True, if something is selected inside cell, or several cells are selected in table. In this case GetSelectionBound 203 can returns -1s.

In editor, you can use TopLevelEditor.GetSelectionBounds to get information about selection in the edited cell.

Hypertext 35

Hypertext items can be not only in main document, but in subdocuments (cells 567 or cell inplace-editor).

Use GetJumpPointLocation 197 instead of GetJumpPointItemNo 196 to access hypertext item.

Hypertext links are numbered sequentially through the document, including all hyperlinks cells. This number is passed as **id** parameter in hypertext events

Checkpoints 30

Unlike hypertext links, each subdocument (main document, table cell) has its own list of *checkpoints*. So methods for *checkpoints* enumeration cannot find *checkpoints* in table cell (if called for the main document).

Page Breaks

Mandatory page breaks inside tables are not supported yet.

1.3.3 Table Cells

Table Cells

The main property of a table is Cells 537[<Row>,<Col>].

<Row>,<Col> are indices of a row and a column, from 0.

Attention: The standard Delphi grids use Cells[<Col>,<Row>] instead!

The type of Cell[r,c] is TRVTableCellData 567. Each RichView control has a similar class inside (richview.RVData 151). Cell and richview.RVData are descendants from the same class 608 designed to store formatted RichView document.

Cells also have additional table-related properties (discussed below).

Table Rows

Tables have another property: Rows 543.

It is a list of table rows. Each row is a list of cells.

Each row has the same number of cells (**important!**)

So:

- number of rows in table = table.Rows.Count;
- number of columns in table = table.Rows[0].Count (each row has the same number of cells as the 0-th)
- table.Cells[r,c] is the same as table.Rows[r][c].

Accessing and Modifying Cells

In editor, you need no special processing to access cells.

Methods working with current/selected item(s) do it automatically (see more info [112]).

Cells have the most of the methods which RichView (not RichViewEdit!) has.

For example:

```
with table.Cells[r,c] do begin
  Clear;
  AddNLATag('hello!', 0, 0, '');
end;
```

Read more about methods of cell [576].

Initially every cell contains one empty line of the 0-th text and paragraph style.

Cells do not have editor-style methods. RichViewEdit creates special inplace editor to edit cell.

Read more about cell editing [117].

1.3.4 Colors and Layout of Tables

Table Colors

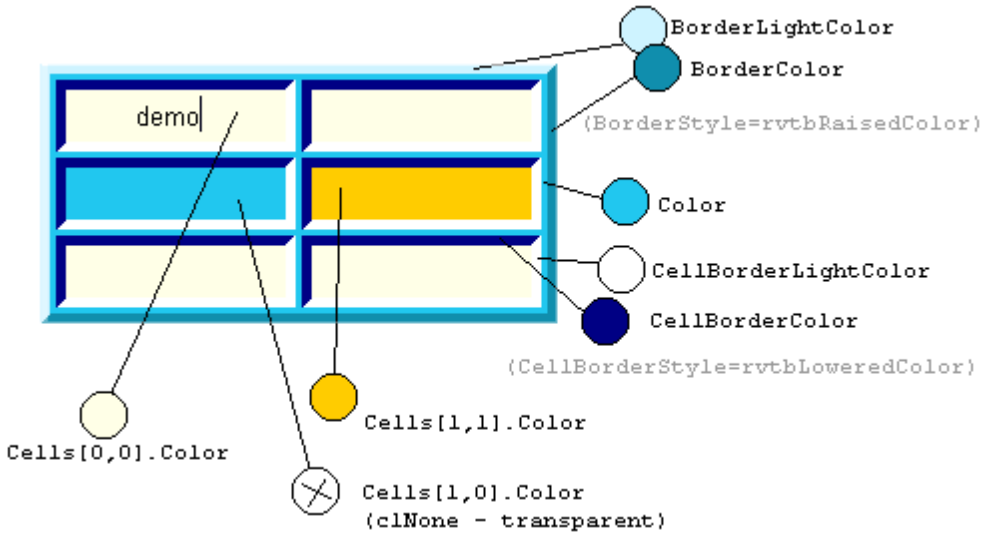


Table Colors

The main color properties of tables (Color [538], BorderColor [532], BorderLightColor [532], CellBorderColor [534], CellBorderLightColor [534]) are shown on the figure above.

There are some more color properties related to rules (lines between cells), see below [115].

Table can be transparent, if its Color [538] is set to *clNone*, or semitransparent [540].

Cells can be individually colored [570], with specified opacity [571]. You can specify default colors for groups of rows and columns [546].

Borders can be hidden, if their widths are set to 0 (see BorderWidth [534] and CellBorderWidth [535]). These are default values.

Cell borders can be 3d, like on this figure, or flat (see BorderStyle [533] and CellBorderStyle [535]).

You can hide some sides of borders of some cells (see cells' VisibleBorders[575]) and set individual color for cell border (cell.BorderColor[569] and cell.BorderLightColor[570]).

When a table with invisible border is inserted in TRichViewEdit[282], such borders are shown as dotted gray lines (see also table Options[540]).

Background Images

Tables and cells can have background images.

See properties of table: BackgroundImage[530], BackgroundStyle[531].

See properties of cell: BackgroundImage[567], BackgroundStyle[568].

Borders and Spacing

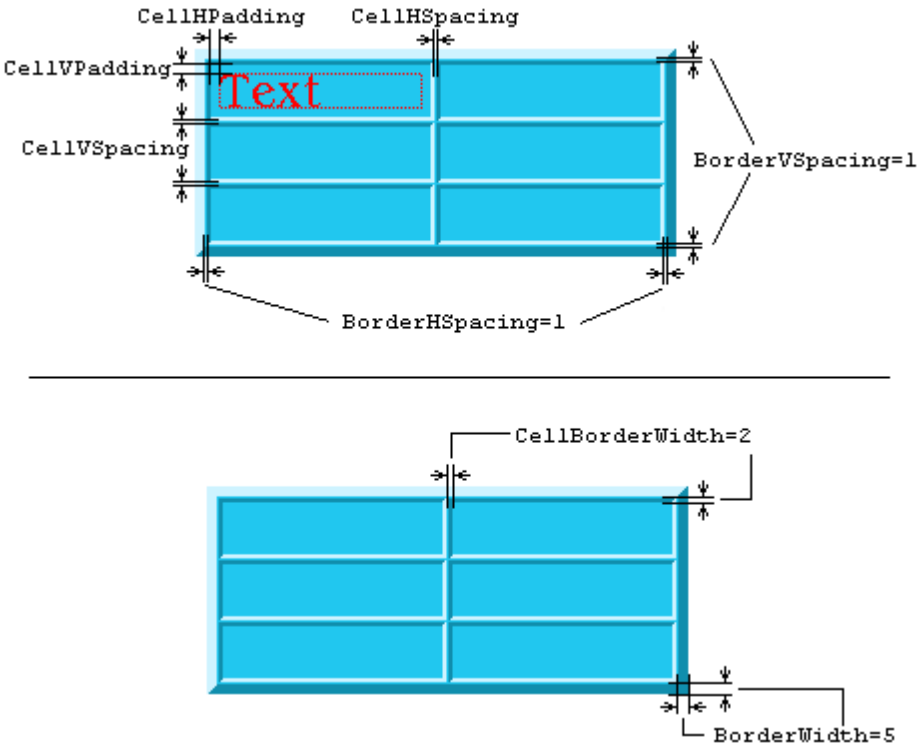


Table Borders

The figure above shows properties affecting the table layout:

- CellHSpacing[536], CellVSpacing[537],
- BorderHSpacing[532], BorderVSpacing[533],
- BorderWidth[534],
- CellBorderWidth[535],
- CellHPadding[536], CellVPadding[537].

(The dotted red line around the cell contents is imaginary).

Rules

Rules are horizontal and/or vertical lines between cells, and may be also between table borders and outermost cells.

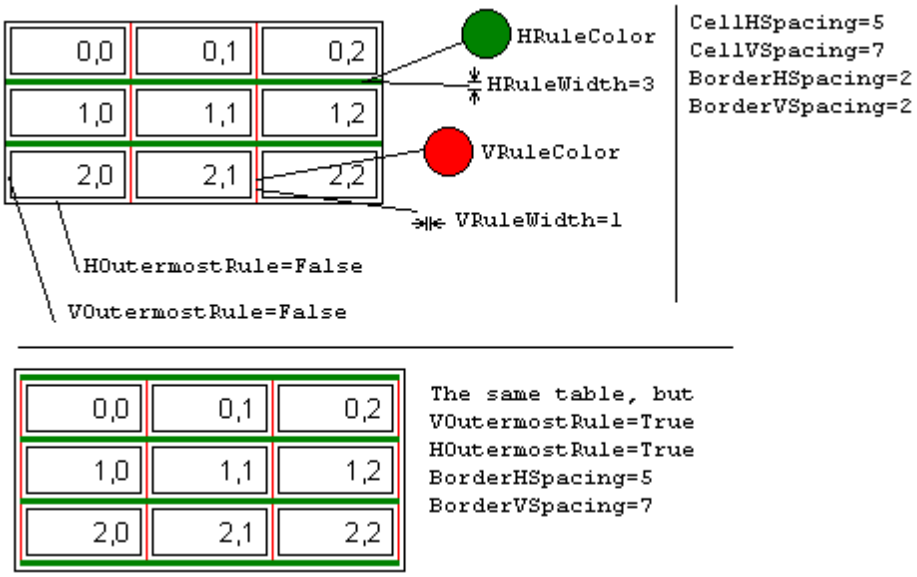


Table Rules

By default rules are invisible.

Rules are defined with properties: HRuleWidth[540], HRuleColor[539], HOutermostRule[539], VRuleWidth[545], VRuleColor[545], VOutermostRule[545].

Tip: How to make 1 pixel border

RichView tables have HTML-style layout, so making one-pixel width borders is a trick.

Even if you set CellBorderWidth[535]=1, CellHSpacing[536]=CellVSpacing[537]=0, 1-pixel width borders of adjacent cells will be visually combined in 2-pixel width borders.

Recommended solution:

setting CellBorderWidth[535]=1, CellHSpacing[536]=CellVSpacing[537]=-1. These values are specially supported by SaveHTMLEx[217] (and SaveHTML[216] with *rvsoForceNonTextCSS* in the Options parameter).

This example assumes RVStyle.Units[404]=*rvstuPixels*. For rvstuTwips, use 15 instead of 1, and -15 instead of -1.

Other solutions (not recommended):

- hiding cells border and using rules; disadvantages: rules cannot be saved in HTML and RTF;
- using VisibleBorders[575] to hide one of adjacent border sides

1.3.5 Selection in Table

Types of Selection

Table can be:

- not selected or
- selected completely (as a part of larger selection) or
- selected partially (multicell selection) or
- some part of cell content (in cell or in inplace editor) selected.

RichView does not support selection started somewhere else and finished at the middle of a table. Selection containing several items can contain tables only completely (or it does not contain them).

Any method working with selection must be called only when the document is formatted.

Tables in Larger Selection

To determine if the table is selected completely, as a part of larger selection, use RichView.GetSelectionBounds 203 (and compare its results with GetItemNo 194 for table).

Multicell Selection

Table can be selected partially: selection can starts on one cell of table, and ends on another cell.

You can determine if the table selected partially using the methods TRVTableItemInfo.GetSelectionBounds 553 or GetNormalizedSelectionBounds 553 (the latter is more convenient).

You can select some cells in table using TRVTableItemInfo.Select 558 and remove the selection using TRVTableItemInfo.Deselect 551.

You can select some rows or columns completely: TRVTableItemInfo.SelectRows 558 and SelectCols 558.

You can determine if the given cell is selected or not with the method IsCellSelected 556.

Selection Inside Cell

You can get inplace-editor and position for currently edited cell: GetEditedCell 552.

See also: TRichViewEdit TopLevelEditor 295.

Before making selection inside cells, you need to activate inplace editor for it (TRichView does not have inplace editors, but the same code must be called in TRichView too, not only in TRichViewEdit).

Example 1 (selecting table.Cells[0,0]):

```
table.EditCell(0,0);
table.Cells[0,0].GetRVData.SelectAll;
```

Example 2 (does the same)

```
uses CRVData, CRVFData;
var RVData: TCustomRVData;
...
RVData := table.Cells[0,0];
RVData := RVData.Edit;
TCustomRVFormattedData(RVData).SelectAll;
```

The method in the second example (calling RVData := RVData.Edit, then selecting) can be used for any type of RVData, not only for table cells.

Editing Operations

Some RichViewEdit 282 methods can be applied to the selected cells (regardless the type of selection):

- ApplyTextStyle 300
- ApplyParaStyle 297
- ApplyStyleConversion 299
- ApplyParaStyleConversion 298
- DeleteSelection 306.

1.3.6 Editing Cells

Editing Cells

Then the user clicks some cell in table with the mouse, or enter to some cell using keyboard, RichViewEdit creates a special editor for this cell. This editor is referred in this documentation as cell inplace editor.

(this editor will be created only if *rvtoEditing* is in table Options 540)

This editor does not have borders and located in place of cell, creating an illusion of editing directly in main

RichViewEdit.

Inplace editing is possible only when table was inserted in TRichViewEdit[282], not in TRichView[128].

When inplace editor is initiated, RichViewEdit moves all content of the cell to this editor. When editor is destroyed, RichViewEdit moves modified data back to the cell (this procedure is transparent for user).

So if you wish to access directly to cell contents[576], use methods of cell's GetRVData[575] instead of methods of the cell itself.

You can initiate editing of cell from code: table.EditCell[552] (see the example in the topic about selection in tables[116]) or Cell.Edit[610].

Position of edited cell can be returned by method GetEditedCell[552] and GetNormalizedSelectionBounds[553].

Limitations

This mechanism of cell editing adds some limitations:

You must not destroy inplace editor from inside the following events: OnKeyDown, OnKeyUp, OnKeyPress, OnRVMouseMove[254], OnStartDrag, OnDragOver.

Inplace editor can be destroyed by calling RichViewEdit.Clear[175], or by the methods placing caret/selection outside the editing cell (RichViewEdit.SetSelectionBounds[235], table.EditCell[552], etc.). You must avoid calling these methods in these events.

If you need to perform such operation, use the following technique:

- from the event, post a window message to the form (PostMessage)
- in the handler procedure for this message, perform this operation.

Example 1 (reloading on double click)

Define some constant > WM_USER:

```
const WM_RELOADDOC = WM_USER+5;
```

In the form's declaration:

```
procedure WMReloadDoc(var Msg: TMessage); message WM_RELOADDOC;
```

Implementation:

```
procedure TForm1.WMReloadDoc(var Msg: TMessage);
begin
  { code for loading here }
end;
```

Double click:

```
procedure TForm3.RichViewEdit1DblClick(Sender: TObject);
begin
  PostMessage(Handle, WM_RELOADDOC, 0, 0);
end;
```

Example 2:

- **Demos\Delphi\Assorted\Graphics\DragImg\ or**
- **Demos\CBuilder\Assorted\Graphics\DragImg**

1.3.7 Cell Merging

Cells have read-only ColSpan[570] and RowSpan[574] properties, similar to html <td colspan rowspan>, 1 by default.

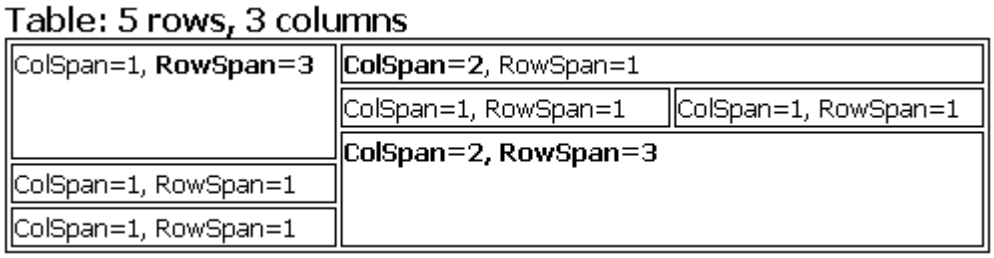
If Cells[537][r,c].ColSpan>1, this cell spans over (ColSpan-1) cells to the right:

Cell[r,c], Cell[r,c+1], ..., Cell[r,c+Cell[r,c].ColSpan-1].

In this case, all overlapped cells (Cell[r,c+1], ..., Cell[r,c+Cell[r,c].ColSpan-1]) are destroyed and equal to nil.

Number of cells in the row is still equal to the number of cells in all other rows, even with merging.

The same is with RowSpan: cell with RowSpan>1 spans over (RowSpan-1) cells below.



ColSpan and RowSpan example

Example: iterating through all cells of table:

```
for r := 0 to table.Rows[543].Count-1 do
 for c := 0 to table.Rows[543][r].Count-1 do
   if table.Cells[537][r,c]<>nil then
     table.Cells[537][r,c].Color[570] := clRed;
// ( Cells[r,c]=nil if it is overlapped by the top left cell)
```

You can get the top left cell which overlaps given cell with table.Rows[543].GetMainCell method:

```
function GetMainCell[579](ARow,ACol: Integer;
  var MRow, MCol: Integer): TRVTableCellData;
```

Methods for merging cells:

- MergeCells[556],
- MergeSelectedCells[556]

Inverse operation:

- UnmergeCells[564],
- UnmergeSelectedCells[564]

Splitting cells:

- SplitSelectedCellsHorizontally[563],
- SplitSelectedCellsVertically[563]

1.3.8 Table Operations

Operations on Selection[116]

Inserting rows and columns:

- InsertColsLeft[554],
- InsertColsRight[554],
- InsertRowsAbove[555],
- InsertRowsBelow[555].

Deleting rows and columns:

- DeleteSelectedCols[551],
- DeleteSelectedRows[551],

Cell merging and unmerging[118]:

- MergeSelectedCells[557] (see also CanMergeSelectedCells[548]),
- UnmergeSelectedCells[564],
- SplitSelectedCellsHorizontally[563],
- SplitSelectedCellsVertically[563]

Other Operations

Inserting rows and columns:

- InsertCols 554,
- InsertRows 555.

Deleting rows and columns:

- DeleteCols 549,
- DeleteRows 550.

Cell merging and unmerging 118**:**

- MergeCells 556 (also CanMergeCells 547)
- UnmergeCells 564,
- DeleteEmptyRows 550, DeleteEmptyCols 550 (use after cell merging).

Assigning Table Properties (As Editing Operations)

Methods for assigning table properties:

- SetTableVisibleBorders 563.

Methods for assigning row properties:

- SetRowVAlign, SetRowKeepTogether, SetRowPageBreakBefore 562.

Methods for assigning cell properties:

- SetCellColor 560, SetCellBorderColor 560, SetCellBorderLightColor 560;
- SetCellVAlign 562;
- SetCellRotation 561;
- SetCellTag 562;
- SetCellBestWidth 560, SetCellBestHeight 559;
- SetCellBackgroundImage 559, SetCellBackgroundImageFileName 559, SetCellBackgroundStyle 559;
- SetCellHint 561;
- SetCellVisibleBorders 562;
- SetCellOpacity 561;
- SetCellOptions 561.

Operations in Viewer and Editor

If the operations above are performed before inserting the table, no addition actions are required.

If these operations are performed on table inserted in TRichView 128, you need to call Format 182 method after to update the document view.

If these operations are performed on table inserted in TRichViewEdit 282, a special sequence of steps is required:

1. checking RichViewEdit.CanChange 302 (especially important for data-aware versions of components);
2. obtaining table object and its position in editor;
   (position is defined as item index + editor where this table is inserted (can be root editor or cell inplace-editor, referred below as rve)
3. calling rve.BeginItemModify 301 for table
4. performing operations
5. calling rve.EndItemModify 307
6. calling rve.Change 303

The example is below.

You can group several actions so that they will be undone/redone as whole, using SetUndoGroupMode 355

In order to perform operation(s) on table, you need to get table object.You can do it using method

```
function TRichView.GetItem 190(ItemNo: Integer): TCustomRVItemInfo;
```

TCustomRVItemInfo 526 is an ancestor class for all items of RichView, including table (TRVTableItemInfo 528). This method can be used for item of any type, so you need to check if it is a table (using "is" operator, or checking RichView.GetItemStyle 194(ItemNo)=rvsTable)

But usually, when you need to perform operations on the item at the position of caret in editor, you do not know if the current item is in "root" editor, or inside cell, or inside cell of table inside other cell, and so on.

In any case, you can get item in position of caret with method

```
function TRichViewEdit.GetCurrentItem[311]: TCustomRVItemInfo;
```

But even if the caret is inside table, the current item will be not a table, but some other item in cell-inplace editor!

The problem can be solved with the method

```
function TRichViewEdit.GetCurrentItemEx[311](
  RequiredClass: TCustomRVItemInfoClass;
  var ItemRichViewEdit: TCustomRichViewEdit;
  var Item: TCustomRVItemInfo): Boolean;
```

In this method,

- **RequiredClass** – set it to TRVTableItemInfo (for C++Builder: __classid(TRVTableItemInfo)).
- **Return value**: True if there is item of the given class at the position of caret (i.e. the caret is to the left or to the right of table), or if the caret is inside item the given class (i.e. the caret is in table cell).
- **Item** receives the top-level item of the given class (i.e. table).
- **ItemRichViewEdit** receives the parent editor for **Item** (this **Item** is in this editor).

This editor can be "root" RichViewEdit or cell inplace editor.

Example

```
// MyRichViewEdit:TRichViewEdit is an editor
// placed on the form at design time.
// Note: the most of operations are performed in
// rve (editor returned by GetCurrentItemEx),
// not in MyRichViewEdit.
var item: TCustomRVItemInfo[526];
    table: TRVTableItemInfo[528];
    Data: Integer;
    rve: TCustomRichViewEdit[282];
    ItemNo: Integer;
begin
  if not MyRichViewEdit.CanChange[302] or
     not RichViewEdit1.GetCurrentItemEx[311](TRVTableItemInfo, rve, item) then
    exit;
  table := TRVTableItemInfo(item);
  ItemNo := rve.GetItemNo[194](table);
  rve.BeginItemModify[301](ItemNo, Data);
  // performing some operation, for example
  // table.InsertRowsBelow(1);
  // or table.CellPadding := 10
  rve.EndItemModify[307](ItemNo, Data);
  rve.Change[303];
end;
```

1.3.9 Undo/redo in Tables

Undo and Redo

All table operations (adding/deleting rows, merging/unmerging, assignment of table properties) can be undone/redone if table was inserted into RichViewEdit[282].

Operations will not be undone if:

- table was created, but not inserted yet;
- table was inserted in RichView[128];

- table was inserted in RichViewEdit with UndoLimit[295]=0.

Example 1

Two operations: (inserting table and changing its color) will be undone in two steps

```
table := TRVTableItemInfo.CreateEx(4,3, RichViewEdit1.RVData);
if RichViewEdit1.InsertItem('', table) then
  table.Color := clRed;
```

Example 2

One operation (inserting red table) will be undone in one step

```
table := TRVTableItemInfo.CreateEx(4,3, RichViewEdit1.RVData);
table.Color := clRed; // table is not inserted yet
RichViewEdit1.InsertItem('', table);
```

About Using Non-Editor-Style Operations in Editor for Modifying Documents

(AddNL[167], for example)

These methods must not be mixed with editor-style methods (such as InsertText[327]) because they will conflict with undo mechanism. Result of undo-redo operations is unpredictable if non-editor operations were used after editor-style one.

This rule relates not only to contents of the main document, but to cells also.

There are some cases when non-editor-style methods may be useful in editor:

- After Clear[175] method, before any editor-style methods (for generating a document before editing). Clear method clears undo and redo buffers.
- In table cells before inserting a table.
- if UndoLimit[295] of editor = 0 (undo mechanism turned off). But Format[182] must be called before calling any editor-style method.
- After ClearUndo[305] method. But Format[182] must be called before calling any editor-style method.

Undo of Assignment to Properties

Assigning values to cell properties (and to VAlign[578] property of rows) will not be undone.

But there are some special methods.

```
table.Cells[r,c].Color := clRed; // will not be undone
table.SetCellColor(clRed, r,c);  // will be undone.
```

(the complete list of these methods is below)

This was made because of efficiency reasons.

These methods do not update editor view. Use them together with BeginItemModify[301]/EndItemModify[307].

For example

```
var item: TCustomRVItemInfo;
    table: TRVTableItemInfo;
    Data: Integer;
    rve: TCustomRichViewEdit;
    ItemNo: Integer;
begin
  if not RichViewEdit1.CanChange or
     not RichViewEdit1.GetCurrentItemEx(TRVTableItemInfo, rve, item) then
    exit;
  table := TRVTableItemInfo(item);
  ItemNo := rve.GetItemNo(table);
  rve.BeginItemModify(ItemNo, Data);
  table.SetRowVAlign(rvcBottom, 1);
  table.SetCellColor(clRed,1,1);
```

```
  rve.EndItemModify(ItemNo, Data);
  rve.Change;
end;
```

Assignments to all table properties can be undone (but still the sequence BeginItemModify-operations-EndItemModify-Change is required). The only exception is VisibleBorders property, use SetTableVisibleBorders if you want to implement an undoable operation.

Direct assignment to cells and rows properties cannot be undone, so use the methods below to implement an undoable operation.

Methods for assigning table properties:

- SetTableVisibleBorders.

Methods for assigning row properties:

- SetRowVAlign.

Methods for assigning cell properties:

- SetCellColor, SetCellBorderColor, SetCellBorderLightColor;
- SetCellVAlign;
- SetCellBestWidth, SetCellBestHeight;
- SetCellBackgroundImage, SetCellBackgroundImageFileName, SetCellBackgroundStyle;
- SetCellHint;
- SetCellVisibleBorders.

1.3.10 Export of Tables

Tables can be exported with document in the following formats:

- text (including Unicode),
- HTML,
- RTF.

Text Export

Tables have the properties:

- TextRowSeparator – string to write after each row;
- TextColSeparator – string to write between cells.

By default, the both of them are equal to CR+LF (#13#10).

MS Word 2000+ saves tables as text in the same way.

You can set TextColSeparator to TAB character (#09), if you wish, but do not forget that there can be CR+LFs inside cells.

RTF and DocX

RichView saves tables in RTF according to the MS Word 2000+ standard. MS Word 2000 introduced many new RTF keywords to store HTML-like formatting.

Similar keywords are saved in DocX files.

MS Word supports two algorithms of calculating widths of columns: fixed and fit-by-content. Initial widths of table columns specified in RTF/DocX (called "table grid") are adjusted by MS Word according to the specified algorithm. The algorithm takes contents and widths of cells into account.

By default, TRichView saves tables as fixed tables. Generally, MS Word's fixed tables look closer to TRichView tables; however, there are some problems. For example, if we save exact values of Cells[r,c].BestWidth (specified in twips or pixels), MS Word sets width of these cells equal to these values exactly, so a wide content may be truncated. What's why TRichView does not save these values to RTF/DocX. So, after saving, cells having BestWidth = 0 and BestWidth > 0 are undistinguished, their width is taken from a table grid (which is saved according to table size in TRichView). You can turn on saving cells' BestWidth by including

rvtoRTFSaveCellPixelBestWidth to table.Options[540]; however, this is not recommended for fixed tables.

If you include *rvtoRTFAllowAutofit* in table.Options[540], this table will be saved as fit-by-content table (but only if table.BestWidth[531]=0). For such tables, you can safely include *rvtoRTFSaveCellPixelBestWidth* to table.Options[540].

When saving a table grid to RTF/DocX, TRichView uses the current table size. What's why TRichView must be formatted before saving to RTF/DocX, if this document contains tables.

HTML

Some properties can be saved only with CSS (with SaveHTMLEx[217], or SaveHTML[216] with *rvsoForceNonTextCSS* in Options parameter). See comments for table properties.

See Also...

RichView methods for saving and loading[58]

1.3.11 Table Resizing

Table Layout Algorithm

Tables do not have column widths and row heights specified explicitly.

Widths are calculated basing on:

- minimal width required to display document inside cell (if *rvtoIgnoreContentWidth* is not in table.Options[540])
- table.BestWidth[531]
- cells BestWidth[569]

Heights are calculated basing on:

- heights of cells content (if *rvtoIgnoreContentHeight* is not in table.Options[540])
- cells BestHeight[568]

Rules for calculations table width:

1. cells always have widths large enough to contain the widest of their item (if *rvtoIgnoreContentWidth* is not in table.Options[540]);
2. if table.BestWidth[531]<>0, it defines width of the table (but table can be wider because of the rule 1)
3. if table.BestWidth[531]=0 then
    - if all columns have cells with BestWidth[569]s in TRVStyleUnits[644], i.e. pixels or twips (and do not have cells with BestWidth[569]s in %), width of table is calculated basing on cells BestWidth[569]s and minimal widths required to show cells contents without overlapping (in this case mouse resizing works the most similar to MS Word)
    - otherwise, width of table is the same as with BestWidth[531]=-100 (100%) (in this version RichView cannot calculate width of table basing on optimal widths of cells)

Rules for calculating widths of cells are quite complicated.

The most important rules (when widths of cells are conflicting):

- widths of cells to the left have higher priority than widths of cells to the right;
- widths of cells in % have higher priority than widths in pixels/twips.

Resizing with Mouse

Resizing with mouse is possible only in editor.

If you include *rvtoRowSizing* and *rvtoColSizing* in table.Options[540], users will not be able to resize it.

When user resizes table with mouse, he/she actually changes values of BestWidth[569] or BestHeight[568]

properties of cells.

When resizing columns with mouse, and columns to the left and to the right do not have cells with BestWidth[569]s in %, RichViewEdit adjusts BestWidth[569]s of cells of column to the right and to the left, trying to keep their total width unchanged.

When resizing with pressed Shift key, RichView sets only BestWidth[569]s of the column to the left.

1.3.12 Example: table to text

This example "removes" the table at the caret position without removing its content.

This is an editing operation, it can be undone[54] by the user.

```
procedure ConvertTableToText(RichViewEdit: TCustomRichViewEdit[282]);
var table: TRVTableItemInfo[528];
    rv: TRichView[128];
    rve: TCustomRichViewEdit[282];
    Stream: TMemoryStream;
    ItemNo,r,c: Integer;
begin
  if not RichViewEdit.GetCurrentItemEx[311](TRVTableItemInfo[528], rve, TCustomRVItemInfo[526](ta
    exit;
  rv := TRichView[128].Create[179](nil);
  try
    rv.Visible := False;
    rv.Parent := RichViewEdit.Parent;
    rv.Style[156] := RichViewEdit.Style[156];
    rv.RVFTextStylesReadMode[154] := rvf_sIgnore;
    rv.RVFParaStylesReadMode[154] := rvf_sIgnore;
    for r := 0 to table.RowCount[543]-1 do
      for c := 0 to table.ColCount[538]-1 do
        if table.Cells[537][r,c]<>nil then begin
          Stream := TMemoryStream.Create;
          try
            table.Cells[537][r,c].GetRVData[575].SaveRVFToStream[220](Stream, False, clNone, nil,
            Stream.Position := 0;
            rv.InsertRVFFromStream[206](Stream, rv.ItemCount[144])
          finally
            Stream.Free;
          end;
        end;
    ItemNo := table.GetMyItemNo; // like rve.GetItemNo[194](table), but faster
    Stream := TMemoryStream.Create;
    try
      rv.SaveRVFToStream[220](Stream, False);
      Stream.Position := 0;
      rve.SetSelectionBounds[235](ItemNo, 0, ItemNo, 1);
      rve.InsertRVFFromStreamEd[323](Stream);
    finally
      Stream.Free;
    end;
  finally
    rv.Free;
  end;
end;
```

The same for C++Builder:

```
void ConvertTableToText(TCustomRichViewEdit* RichViewEdit)
{
  TCustomRVItemInfo   *item;
  TCustomRichViewEdit *rve;
  if (! RichViewEdit->GetCurrentItemEx(__classid(TRVTableItemInfo), rve, item))
    return;
  TRVTableItemInfo*table = (TRVTableItemInfo*)item;
  TRichView* rv = new TRichView((TComponent*)NULL);
  rv->Visible = false;
  rv->Parent  = RichViewEdit->Parent;
  rv->Style   = RichViewEdit->Style;
  rv->RVFTextStylesReadMode = rvf_sIgnore;
  rv->RVFParaStylesReadMode = rvf_sIgnore;
  for (int r=0; r<table->RowCount; r++)
    for (int c=0; c<table->ColCount; c++)
      if (table->Cells[r][c])
      {
        TMemoryStream*Stream = new TMemoryStream;
        table->Cells[r][c]->GetRVData()->SaveRVFToStream(Stream, false, clNone, NULL, NULL)
        Stream->Position = 0;
        rv->InsertRVFFromStream(Stream, rv->ItemCount);
        delete Stream;
      }
  int ItemNo = table->GetMyItemNo();
  TMemoryStream*Stream = new TMemoryStream;
  rv->SaveRVFToStream(Stream, false);
  Stream->Position = 0;
  rve->SetSelectionBounds(ItemNo, 0, ItemNo, 1);
  rve->InsertRVFFromStreamEd(Stream);
  delete Stream;
  delete rv;
}
```

1.3.13 Example: dividing table

```
function SplitTable(rve: TCustomRichViewEdit[282]; table: TRVTableItemInfo[528];
  RowCountInFirstTable: Integer; PageBreak: Boolean): Boolean;
var r, c, mr, mc: Integer;
    newtable: TRVTableItemInfo[528];
    Stream: TMemoryStream;
begin
  Result := False;
  if RowCountInFirstTable>=table.RowCount[543] then
    exit;
  for c := 0 to table.ColCount[538]-1 do
    if table.Cells[537][RowCountInFirstTable, c] = nil then begin
      table.Rows[543].GetMainCell[579](RowCountInFirstTable, c, mr, mc);
      if mr<RowCountInFirstTable then
        exit;
    end;
  Stream := TMemoryStream.Create;
  table.SaveRowsToStream[557](Stream, RowCountInFirstTable,
    table.RowCount[543]-RowCountInFirstTable);
  Stream.Position := 0;
  newtable := TRVTableItemInfo[528].CreateEx(0, 0, rve.RVData);
  newtable.LoadFromStream[556](Stream);
```

```
  if PageBreak then
    newtable.PageBreakBefore := True;
  Stream.Free;
  rve.BeginUndoGroup(rvutModifyItem);
  rve.SetUndoGroupMode(True);
  try
    table.DeleteRows(RowCountInFirstTable, table.RowCount-RowCountInFirstTable, True
    // table.GetMyItemNo is the same as rve.GetItemNo(table), but faster
    rve.SetSelectionBounds(table.GetMyItemNo, 1, table.GetMyItemNo, 1);
    rve.InsertItem('', newtable);
  finally
    rve.SetUndoGroupMode(False);
  end;
  Result := True;
end;
```

Parameters:

rve – editor directly containing **table**;

table – the table to divide.

These two parameters are usually a result of RichViewEdit1.GetCurrentItemEx.

The table will be divided into two tables.

The first resulting table will have row count = **RowCountInFirstTable**, the second table will have row count = table.RowCount-**RowCountInFirstTable**.

If **PageBreak** parameter is *True*, a page break will be inserted between tables.

Return value: *True* if the table was divided.

This is an editing operation, it can be undone and redone by the user.

1.4 Components

Components

Icon	Class Name	Description
	TRVStyle	Component for defining visual appearance of RichView, RichViewEdit, DBRichView, DBRichViewEdit. It contains some properties which can be used by several RichViews.
	TRichView	Component for displaying hypertext documents (containing pictures, controls, tables)
	TRichViewEdit	Component for editing hypertext documents (containing pictures, controls, tables)
	TDBRichView	Data-aware version of RichView
	TDBRichViewEdit	Data-aware version of RichViewEdit
	TRVPrint	Component for printing documents contained in the components above
	TRVPrintPreview	Component allowing to show document as it will be printed by RVPrint
	TRVReportHelper	Component for drawing documents on different canvas (screen, printer, image, etc.)

	TRVOfficeConverter 496	Component allowing to use Microsoft Office text converters (import and export in different file formats)

1.4.1 Visual Components

1.4.1.1 TRichView

TRichView is a control for displaying documents with pictures, tables, Delphi controls and hyperlinks.

Unit RichView;

Syntax

```
TCustomRichView = class(TRVScroller 508)
TRichView = class(TCustomRichView)
```

TRichView does not display its content if it is not linked to TRVStyle component via Style 156 property.

The most important property settings can be done in the component editor 134 for TRichView. In Delphi (C++Builder) IDE, right click the RichView object on the form, choose "Settings" in the context menu.

Hierarchy

TObject
TPersistent
TComponent
TControl
TWinControl
TCustomControl
TRVScroller 508
TCustomRichView

List of properties, events and methods

grouped in several categories (some entries can be in several groups)

- General 128
- Background 128
- Appending Items 129
- Hypertext 129
- Style templates 130
- Scrolling, margins, sizes 130
- Selection and Clipboard 130
- Saving and loading as RVF 131
- Other methods for saving and loading 131
- Information about items 132
- Modifying items 132
- Checkpoints 132
- Mouse events 133
- Unicode 133
- Live spelling check 133
- Picture animation 133
- Working in palette mode 133
- Other properties, methods and events 133
- Obsolete properties and methods 134

General properties

- Style 156 – link to RVStyle component defining a lot of properties which can be used in several RichViews;
- Options 146 – some important options;
- ItemCount 144 – number of items in the document.

Properties defining background

- BackgroundBitmap 136;
- BackgroundStyle 136;
- Color 139.

Appending items (Add*** methods)

Main methods for appending items:

- AddNL 167 adds one text item;
- AddBreakEx 160 adds break (horizontal line);
- AddBulletEx 160 adds bullet (image from ImageList);
- AddHotspotEx 164 adds hotspot (image from ImageList - hypertext link);
- AddPictureEx 169 adds picture;
- AddHotPicture 163 adds picture - hypertext link;
- AddControlEx 162 adds Delphi/C++Builder control;
- AddTab 170 adds tabulator;
- AddItem 166 – general method for appending items (usually used for tables 111);
- also AddNamedCheckpointEx 166 – adds checkpoint which will be associated with next added item.

"Tagged" analogs of the methods above (**Add***Tag** methods) (the same methods, but have additional Tag parameter):

- AddNLTag 167;
- AddBreakExTag 160;
- AddBulletExTag 160;
- AddHotspotExTag 164;
- AddPictureExTag 169;
- AddHotPictureTag 163;
- AddControlExTag 162;
- also AddNamedCheckpointExTag 166 and AddNLATag, AddNLWTag 167.

Additional methods for appending items:

- Add 167 adds one text item in last paragraph;
- AddFmt 163 adds one formatted string assembled from a Pascal format string and an array arguments;
- AddTextNL, -A -W 171 add several text items from text, separated by CR+LF characters;
- AddBreak 160 provides a subset of functionality of AddBreakEx 160;
- also: AddCheckpoint 166 provides subset of functionality of AddNamedCheckpointEx 166;
- also Unicode-enabled methods 133.

"Tagged" analogs of the methods above:

- AddTag 167;
- AddBreakTag 160;
- also AddCheckpointTag 166.

Also

- AppendFrom 172 copies content from another RichView;
- InsertRVFFromStream 206,
- AppendRVFFromStream 173.

See also

- Clear 175 clears document;
- SetAddParagraphMode 225;
- Format 182, FormatTail 183 format the document;
- Reformat 214 reformats the document.

Hypertext

- event OnJump 247 occurs when user clicks hyperlink;
- event OnRVMouseMove 254 occurs when mouse pointer moves over hyperlink;
- property FirstJumpNo 144 – use to set hypertext id for the first hypertext item;
- GetJumpPointItemNo 196 (deprecated) returns index of item with the specified hypertext id;
- GetJumpPointLocation 197 should be used instead of GetJumpPointItemNo in applications working with tables 111;
- GetJumpPointY 197 returns Y coordinate of item with the specified hypertext id.

Style templates

Properties:

- UseStyleTemplates[158] allows or disallows using Style[156].StyleTemplates[402].
- StyleTemplateInsertMode[156] specifies how style templates are used when inserting RTF or RVF content.

Events:

- OnStyleTemplatesChange[266] occurs then the collection of style templates has been modified.

Scrolling, margins and sizes

Properties:

- LeftMargin[145], RightMargin[149], TopMargin[158], BottomMargin[137] – margins;
- MaxTextWidth[146] – maximal width for word wrapping;
- MinTextWidth[146] – minimal width for word wrapping (minimal width of scrollable area, not including margins);
- DocumentHeight[142] – height of document (height of scrollable area);
- VAlign[159] – vertical alignment of document;
- VScrollVisible[512] – hides/shows vertical scrollbar;
- HScrollVisible[510] – hides/shows horizontal scrollbar;
- VSmallStep[159] – number of pixels in one step of vertical scrollbar;
- FirstItemVisible[144] – the first visible item;
- LastItemVisible[144] – the last visible item;
- MaxLength[145] – maximal count of characters per line;
- WordWrap[159] allows to disable word wrapping;
- Tracking[511] – whether RichView repaints while its scrollbar thumb tab is dragged with the mouse.

Methods:

- ClientToDocument[176] converts client coordinates to document coordinates

Events:

- OnVScrolled[513] occurs when document is scrolled vertically
- OnHScrolled[513] occurs when document is scrolled horizontally

Also:

- property CPEventKind[139]
- event OnCheckPointVisible[238]
- method GetItemCoords[191]

Methods for working with selection and the Clipboard

- SelectionExists[224] returns "is something selected?"l

- Copy[177] copies selection to the Clipboard in all available formatsl
- CopyDef[177] copies selection to the Clipboard using formats specified in Options[146];
- CopyImage copies selected image to the Clipboard;
- CopyTextA[178] copies selection as text to the Clipboard;
- CopyTextW[178] copies selection to the Clipboard as Unicode text;
- CopyRVF[178] copies selection as RVF to the Clipboard;
- CopyRTF[178] copies selection as RTF (Rich Text Format);

- GetSelectedImage[202] returns selected image
- GetSelText[204] returns selection as text
- GetSelTextW[204] returns selection as Unicode string;

- GetOffsBeforeItem[201], GetOffsAfterItem[200] return offset before/after the given item; useful with SetSelectionBounds[235];

- SelectAll[223] selects the whole document for copying to the Clipboard;
- SetSelectionBounds[235] selects the specified items;

- SelectWordAt 224 selects the word at (X,Y);
- SelectControl 223 selects a visual control;
- Deselect 181 clears selection (not contents, but bounds);

- GetSelectionBounds 203 returns bounds of selection;
- event OnSelect 265 occurs when selection bounds are changed;
- event OnCopy 240 allows to copy selection to the Clipboard in custom formats;
- property SelectionHandlesVisible 156 shows or hides selection handles for a touch screen.

Properties, methods and events for loading and saving RVF

Properties:

- RVFOptions 152 – options for saving and loading RVF;
- RVFWarnings 155 describes warnings or errors occurred during the last reading of RVF.

Events:

- OnRVFControlNeeded 251 requests inserted control when reading RVF, if RVF was saved without controls' "bodies";
- OnRVFImageListNeeded 252 requests ImageList when reading bullet 96 or hotspot 97 from RVF;
- OnRVFPictureNeeded 252 requests image when reading RVF, if RVF was saved without images' "bodies".

Methods:

- SaveRVF 220 saves document or selection as RVF to the file;
- SaveRVFToStream 220 saves document or selection as RVF to the stream;
- LoadRVF 211 loads RVF from the file;
- LoadRVFFromStream 211 loads RVF from the stream;
- CopyRVF 178 copies selection as RVF to the Clipboard;
- InsertRVFFromStream 206 inserts RVF from the stream;
- AppendRVFFromStream 173 appends RVF from the stream using the specified paragraph style for the first RVF item.

Also:

- methods Copy 177, CopyDef 177.

Other saving and loading

Methods:

- LoadFromStream 208 loads document in RVF, RTF or text formats;
- LoadText 212 appends text form the file;
- LoadTextW 212 appends text from the file in Unicode encoding;
- LoadRTF 209, LoadRTFFromStream 210 appends the file or the stream in RTF (Rich Text Format);
- SaveHTML 216, SaveHTMLToStream 218 save the document in an HTML file/stream using a basic HTML syntax;
- SaveHTMLEx 217, SaveHTMLToStreamEx 218 save the document in an HTML or XHTML file/stream using CSS;
- SaveText 221, SaveTextToStream 221 save the document in a file/stream as ANSI text;
- SaveTextW 221, SaveTextToStreamW 221 saves the document in a file/stream as a Unicode text;
- SaveDocX 215, SaveDocXToStream 216 saves the document (or selection) in a file/stream as DocX (Microsoft Word format);
- SaveRTF 219, SaveRTFToStream 220 saves the document (or selection) in a file/stream as RTF.

Properties:

- RTFOptions 150 – options for RTF export.

Events:

- OnSaveComponentToFile 256 allows to save inserted controls in HTML and text files;
- OnWriteHyperlink 267, OnReadHyperlink 248 allow to customize saving/loading of hypertext links in HTML and RTF files;
- OnHTMLSaveImage 241, OnSaveImage2 260 allow to save images with HTML in custom formats and with custom file names;
- OnSaveItemToFile 261 allows to save any item in RTF, HTML or text files in your own format;

- OnImportPicture[242] allows custom processing of external images when importing RTF.

Information about items (Get***Info methods)

- GetItemStyle[194] returns a type of item;

- GetTextInfo[204] – information about text item;
- GetBreakInfo[183] – about "break";
- GetBulletInfo[184] – about "bullet";
- GetHotspotInfo[189] – about "hotspot";
- GetPictureInfo[201] – about picture or "hot-picture";
- GetControlInfo[187] – about inserted control;
- GetItemExtraIntProperty[Ex][193], GetItemExtraStrProperty[Ex][193] return value of extra item property;
- GetItem[190] – general method for obtaining items (usually used for tables[111]);

- GetItemCheckpoint[191] – about item's "checkpoint";
- GetItemVAlign[196] – about item's position relative to line;
- GetItemTag[195] – about item's "tag";
- GetItemText[195] – about text/name of item;
- IsParaStart[208] returns "is this item a first item in paragraph?";
- IsFromNewLine[207] returns "is this item a first item in paragraph?";
- GetItemPara[194] returns an index of paragraph containing a specified item;
- GetItemNo[194] returns item index for object representing item (usually used for tables[111]);
- GetItemCoords[191] returns coordinates of item relative to document area, GetItemClientCoords[191] returns coordinates of item relative to client area of RichView window;
- properties PageBreaksBeforeItems[149], ClearLeft[138], ClearRight[138].

Modifying items (Set***Info methods)

- SetBreakInfo[226] changes *break*[93];
- SetBulletInfo[227] changes *bullet*[96];
- SetHotspotInfo[229] changes *hotspot*[97];
- SetPictureInfo[234] changes picture[90] or *hot-picture*[93];
- SetControlInfo[228] changes inserted control[94];
  also
- SetItemTag[231] changes item's "tag";
- SetItemVAlign[233] – change the item position relative to the line;
- SetItemText[232] changes text/name of item;
- SetItemExtraIntProperty[Ex][230], SetItemExtraStrProperty[Ex][231] sets value of extra item property;
- SetCheckpointInfo[227] sets or changes item's "checkpoint";
- RemoveCheckpoint[214] removes item's "checkpoint";
- see also properties PageBreaksBeforeItems[149], ClearLeft[138], ClearRight[138].

Working with *checkpoints*

Obtaining checkpoint data:

- GetFirstCheckpoint[188], GetNextCheckpoint[199] allow to receive full list of *checkpoints*;
- GetLastCheckpoint[198], GetPrevCheckpoint[202] – the same in backward direction;
- GetCheckPointByNo[185] returns a *checkpoint* with the specified index;
- GetItemCheckpoint[191] returns the item's *checkpoint*;
- event OnCheckPointVisible[238] occurs when *checkpoint* (or "section", marked by this *checkpoint*) becomes visible as a result of vertical scrolling;
- FindCheckPointByName[182] searches for *checkpoint* with the specified name;
- FindCheckPointByTag[182] searches for *checkpoint* with the specified *tag*.

Extracting properties from *checkpoint* data:

- GetCheckpointInfo[185] returns main properties of the *checkpoint*;

- GetCheckpointItemNo 186 returns an index of item which owns the *checkpoint*;
- GetCheckpointNo 186 returns an index of the *checkpoint*;
- GetCheckpointYEx 187, GetCheckpointXY 186 returns coordinates of the *checkpoint*.

Also:

- GetCheckPointY 186 returns Y coordinate of *checkpoint* by its index.

See also methods for obtaining information about items 132 and for modifying document 132.

Mouse events

- OnRVDblClick 250 occurs on double click;
- OnRVMouseDown 253 occurs when user presses mouse button;
- OnRVMouseUp 254 occurs when user releases mouse button;
- OnRVRightClick 255 occurs on right click.
- OnItemHint 246 allows customizing hints displayed when the mouse pointer moves above an item.
- OnGetItemCursor 240 allows customizing cursors displayed when the mouse pointer moves above an item.

See also hypertext events 129.

Live spelling check

- StartLiveSpelling 237 starts live spelling thread;
- ClearLiveSpellingResults 175 clears live spelling underlines and stops the thread;
- LiveSpellingValidateWord 208 removes live spelling underlines from the specified word;
- OnSpellingCheck 265 event – the place to check word spelling.

Picture animation

Properties:

- AnimationMode 135 specifies when image animation 56 starts.

Methods:

- StartAnimation 237 starts all animations;
- StopAnimation 237 stops all animations;
- ResetAnimation 215 rewinds all animations to the first frame.

Unicode (in Delphi 3-2007)

- AddNLATag 167 adds one text item in ANSI encoding;
- AddNLWTag 167 adds one text item in Unicode encoding;
- AddTextNLW 171 adds one or more text items in Unicode encoding;
- LoadTextW 212, LoadTextFromStreamW 213 appends text from file/stream in Unicode encoding;
- SaveTextW 221, SaveTextToStreamW 221 saves Unicode file/stream;
- GetSelTextW 204 returns selection as Unicode string;
- CopyTextW 178 copies selection to the Clipboard as Unicode.

Working in palette mode

- property DoInPaletteMode 142;
- method UpdatePaletteInfo 238.

Others

Properties:

- RVData 151 – object representing document;
- Cursor 140 – cursor;
- Delimiters 140 – characters not belonging to words;
- TabNavigation 157 sets mode for navigation through hypertext links and controls;
- AnimationMode 135 defines when animated images starts animation;
- DocProperties 141 – additional information for saving in RVF 59;
- DocParameters 140 – store document page parameters;

- NoteText 146 – text for references to parent notes.

Methods:

- DeleteUnusedStyles 181 deletes unused text, paragraph and list styles from TRVStyle;
- MarkStylesInUse 213, DeleteMarkedStyles 180 – advanced methods for deleting unused styles;
- SearchText, -A, -W 222 search for the text string;
- GetWordAt 205 returns a word at the specified point;
- DeleteItems 179, DeleteSection 180 delete the specified range of items from document;
- DeleteParas 180 deletes the specified range of paragraphs, updates the document;
- FindControlItemNo 182 returns index of item which owns the specified inserted control 94;
- GetFocusedItem 188 returns the item in the focus;
- GetLineNo 198 returns line number in the specified position;
- BeginOleDrag 174 starts drag&drop.

Events:

- OnItemAction 244;
- OnControlAction 239.

Obsolete properties and methods:

AddBullet 160, AddCenterLine 161, AddControl 162, AddFromNewLine 161, AddHotspot 164, AddNamedCheckPoint 166, AddPicture 169, AllowSelection 135, AddText 171, AddTextFromNewLine 171, LineCount 145, SingleClick 156.

1.4.1.1.1 TRichView : Component editor

Component editor for TCustomRichView (and all inherited components: TRichView 128, TRichViewEdit 282, TDBRichView 373, TDBRichViewEdit 377).

To display the component editor dialog in Delphi (C++Builder) IDE, right click RichView object on the form, choose "Settings" in the popup menu.

The dialog consists of 3 pages: Styles, Tags, RVF.

Styles

This page allows to select one of 3 style modes:

- Use a predefined set of styles.
- Allow adding styles dynamically.
- Allow adding styles dynamically + style templates

This setting changes the properties: RVFTextStylesReadMode 154, RVFParaStylesReadMode 154, RVFOptions 152, RTFReadProperties.TextStyleMode 274, .ParaStyleMode 272, UseStyleTemplates 158. Changes are shown in the light yellow box below.

"Use a predefined set of styles" mode

In this mode, the component:

- does not add new items in the collection of text 404, paragraph 398 and list 396 styles of the linked 156 TRVStyle 385 component;
- does not save these styles in RVF 59 files or streams.

"Allow adding styles dynamically" mode

In this mode, the component:

- adds new items in the collection of text 404, paragraph 398 and list 396 styles, if necessary; new styles can be added as a result of RVF and RTF loading;
- saves these styles in RVF 59 files and streams.

If you want to display RTF files close to the original, or use commands for adding styles (for example, inside OnStyleConversion 366 or OnParaStyleConversion 364, or RichViewActions), you must set this mode.

Since version 1.8, "Allow adding styles dynamically" is set by default for all new TRichViews created at design time (placed on forms from the Component Palette), if the defaults were not overridden.

"Allow adding styles dynamically + style templates" mode

All properties are set to the same values as in the previous mode, but UseStyleTemplates[158]=*True*.

Additional information about these modes can be found in the topic "Styles and style templates"[40].

RVF[59]

Defines options for saving and loading RVF (RichView Format)[59] files and strings. These settings change RVFOptions[152].

Storing The Component Editor's Options

Set "Default" checkbox to store settings on the given page. Settings are stored in the Registry. They will be applied to new components created in Delphi/C++Builder IDE from the Component Palette. Stored settings do not affect component creation at run time (from your code), and components that already were added on forms.

1.4.1.1.2 Properties

1.4.1.1.2.1 TRichView.AllowSelection

```
property AllowSelection: Boolean;
```

This property still works, but obsolete, please use *rvoAllowSelection* in Options[146]

1.4.1.1.2.2 TRichView.AnimationMode

Specifies when image animation starts.

```
type
  TRVAnimationMode = (rvaniDisabled, rvaniManualStart, rvaniOnFormat);
property AnimationMode: TRVAnimationMode;
```

(introduced in version 10)

Mode	Meaning
rvaniDisabled	Animation is completely disabled. If images are added/updated in this mode, they cannot be animated (even if different mode is set later). This mode saves system resources.
rvaniManualStart	Animation starts when calling StartAnimation[237] and stops when calling StopAnimation[237].
rvaniOnFormat	Animation starts when calling Format[182] and stops when calling Clear[175]. StartAnimation[237] and StopAnimation[237] work as well.

Default value:

rvaniManualStart

See also:

- Animation in TRichView[56];
- Pictures[90];
- *Hot-Pictures*[93].

1.4.1.1.2.3 TRichView .BackgroundBitmap

Background image shown below the document

```
property BackgroundBitmap: TBitmap;
```

Assignment to this property copies the image, so you still need to free the source image yourself.

Example 1 (assigning image)

```
bmp := TBitmap.Create;
bmp.LoadFromFile('c:\image.bmp');
MyRichView.BackgroundBitmap := bmp;
bmp.Free;
```

Example 2 (doing the same work)

```
MyRichView.BackgroundBitmap.LoadFromFile('c:\image.bmp');
```

Example 3 (clearing the image)

```
MyRichView.BackgroundBitmap := nil;
```

In TRichViewEdit[282], you can change this property as an editing operation, see TRichViewEdit.SetBackgroundImageEd[338].

See also properties:

- BackgroundStyle[136].

1.4.1.1.2.4 TRichView .BackgroundStyle

Defines how BackgroundBitmap[136] is displayed

```
type
  TBackgroundStyle = (bsNoBitmap, bsStretched,
    bsTiled, bsTiledAndScrolled, bsCentered,
    bsTopLeft, bsTopRight,
    bsBottomLeft, bsBottomRight);

property BackgroundStyle : TBackgroundStyle;
```

Note: TBackgroundStyle is declared in RVScroll unit.

Style	Meaning
bsNoBitmap	Colored background (bitmap is not used)
bsStretched	Bitmap is stretched to fit RichView window
bsTiled	Repeats bitmap until the entire RichView window is covered
bsTiledAndScrolled	Repeats bitmap until the entire scrollable area is covered (tiles are scrolled together with content)
bsCentered	Centers bitmap in RichView window
bsTopLeft,	Bitmap is in the top left corner
bsTopRight	Bitmap is in the top right corner
bsBottomLeft	Bitmap is in the bottom left corner
bsBottomRight	Bitmap is in the bottom right corner

In TRichViewEdit[282], you can change this property as an editing operation, see TRichViewEdit.SetIntPropertyEd[337].

See also properties:

- BackgroundBitmap 136;
- Color 139.

See also:

- TRVItemBackgroundType 636 type;
- Smooth image scaling 57.

1.4.1.1.2.5 TRichView.BiDiMode

Specifies the default bidirectional mode for the control.

```
property BiDiMode: TRVBiDiMode 623;
```

(introduced in version 1.6)

Value	Meaning
rvbdUnspecified	Turns off support of right-to-left languages. Bi-directed text will not be processed correctly. Also, bi-di properties will not be saved and loaded from files. Text output is more efficient in this mode, so, if you do not plan working with Arabic or Hebrew texts, you can use this mode.
rvbdLeftToRight	Sets default text flow to left-to-right
rvbdRightToLeft	Sets default text flow to right-to-left; if version of Windows allows it, a vertical scrollbar will be positioned at the left side of the TRichView window.

BiDiMode can be overridden by BiDiMode of paragraph style 450 and BiDiMode of text style 433.

In TRichViewEdit 282, you can change this property as an editing operation, see TRichViewEdit.SetIntPropertyEd 337.

The results depends on the value of Style 156.TextEngine 403:

- if Windows API is selected, in *rvbdUnspecified* mode the results are completely incorrect for RTL text (wrong selection drawing and caret positions); the results are unreliable in other modes (may depend on the chosen font);
- if Uniscribe is selected, the results are correct is all modes; in *rvbdUnspecified* mode, only rearrangement of items on lines is turned off.

See also:

- Bidirectional text in TRichView 69.

Default value:

rvbdUnspecified

1.4.1.1.2.6 TRichView.BottomMargin

Bottom margin of document

```
property BottomMargin: Integer;
```

This value is measured in "screen" pixels, so when printing it's recalculated according to the printer resolution, and margins on pages look like margins on the screen.

In TRichViewEdit[282], you can change this property as an editing operation, see TRichViewEdit.SetIntPropertyEd[337].

See also properties:

- LeftMargin[145];
- RightMargin[149];
- TopMargin[158].

See also properties of TRVPrint:

- Margins[486].

See also:

Additional information about horizontal scrolling, margins, indents, etc.[46]

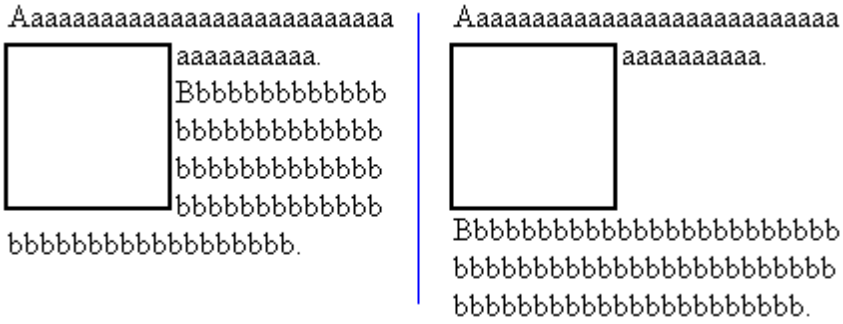
1.4.1.1.2.7 TRichView.ClearLeft

Specifies whether a text flow around left-aligned[648] items should be cleared for the paragraph started from the **Index**-th item.

```
property ClearLeft[Index: Integer]: Boolean
```

(Introduced in version 12)

Set ClearLeft[ItemNo] to *True* if you want the ItemNo-th item to be placed below any left-aligned[648] object.



This property is illustrated on the picture above. At the left side, you can see two paragraphs ("aaa" and "bbb") flow around left-aligned picture. At the right side, you can see how it looks if the first item in the "bbb" paragraph has ClearLeft[] = *True*.

This item must start a new paragraph (IsParaStart[208](ItemNo)) or a new line inside a paragraph (IsFromNewLine[207](ItemNo)), otherwise assignment will be ignored.

You need to reformat RichView after this assignment (using Format[182]).

In editor you can change value of this property as an editing operation, using ClearTextFlow[305] method.

See also:

- ClearRight[138],
- PageBreaksBeforeItems[149],
- TRVVAlign[648] type (*rvvaLeft*).

1.4.1.1.2.8 TRichView.ClearRight

Specifies whether a text flow around right-aligned[648] items should be cleared for the paragraph started from the **Index**-th item.

```
property ClearRight[Index: Integer]: Boolean
```

(Introduced in version 12)

Set ClearRight[ItemNo] to *True* if you want the ItemNo-th item to be placed below any right-aligned[648] object.

This item must start a new paragraph (IsParaStart[208](ItemNo)) or a new line inside a paragraph (IsFromNewLine[207](ItemNo)), otherwise assignment will be ignored.

You need to reformat RichView after this assignment (using Format[182]).

In editor you can change value of this property as an editing operation, using ClearTextFlow[305] method.

See also:

- ClearLeft[138],
- PageBreaksBeforeItems[149],
- TRVVAlign[648] type (*rvvaRight*).

1.4.1.1.2.9 TRichView.Color

Background color

```
property Color: TColor;
```

This property allows to override the default background color specified in RichView.Style[156].Color[387].

RichView paints background with this color, if it is not equal to *clNone*, otherwise RichView uses the default color.

Background color is used if BackgroundBitmap[136] is empty, or BackgroundStyle[136] is not equal to *bsStretched*, *bsTiled* or *bsTiledAndScrolled.*

In TRichViewEdit[282], you can change this property as an editing operation, see TRichViewEdit.SetIntPropertyEd[337].

Default value:

clNone

See also properties:

- BackgroundStyle[136];
- BackgroundBitmap[136].

See also properties of TRVStyle:

- Color[387].

1.4.1.1.2.10 TRichView.CPEventKind

Specifies the mode of generating OnCheckpointVisible[238] event.

```
type
  TCPEventKind =
   (cpeNone, cpeAsSectionStart, cpeWhenVisible);

property CPEventKind: TCPEventKind;
```

Note: TCPEventKind is declared in RVScroll unit.

Value	Meaning
cpeNone	OnCheckpointVisible[238] does not occur
cpeAsSectionStart	OnCheckpointVisible[238] is called for visible *checkpoint* or the nearest *checkpoint* above the visible area (so that the last event is always called for *checkpoint* located in or above the visible area)
cpeWhenVisible	OnCheckpointVisible[238] is generated for visible *checkpoint*

This event occurs only for *checkpoints* having "raise event" flag.

This event never occurs in TRichViewEdit (in this version)

See also events:

- OnCheckpointVisible 238.

See also:

- *Checkpoints* 30 for details.

1.4.1.1.2.11 TRichView.Cursor

Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.

```
property Cursor: TCursor;
```

Default value:

- *crDefault* for TRichView;
- *crIBeam* for TRichViewEdit 282.

See also:

- RichViews' cursors 77.

1.4.1.1.2.12 TRichView.Delimiters

Set of delimiters for specifying a "word" in document

```
property Delimiters: String;
```

Each character of this string is one delimiter.

Unicode note: in Delphi/C++Builder 2009 or newer, this property contains Unicode text. For older versions, it contains ANSI text.

Default value (changed in version 10)

```
' .;,:(){}"/\<>!?[]'#$91#$92#$93#$94'-+*='#$A0#$84
```

See also methods:

- SelectWordAt 224;
- GetWordAt 205;
- SearchText 222.

See also events:

- OnRVDblClick 250;
- OnRVRightClick 255.

See also methods of TRichViewEdit:

- SelectCurrentWord 337.

See also:

- Live spelling check 69.

1.4.1.1.2.13 TRichView.DocParameters

Page properties.

```
property DocParameters: TRVDocParameters 277;
```

(introduced in version 10)

Subproperties of this property are not used by TRichView itself, but can be written and read in RVF 59 and RTF.

Besides, they can be assigned to TRVPrint 479 using the method AssignDocParameters 488.

RVF

DocParameters are stored with DocProperties [141].

How to save to RVF: include *rvfoSaveDocProperties* in RVFOptions [152].

How to load from RVF: include *rvfoLoadDocProperties* in RVFOptions [152].

RTF and DocX

How to save to RTF/DocX: include *rvrtfSaveDocParameters* in RTFOptions [150].

How to load from RTF: set RTFReadProperties [151].ReadDocParameters [273] = *True*.

See also events:

- OnSaveRTFExtra [263];
- OnSaveDocXExtra [257].

1.4.1.1.2.14 TRichView.DocProperties

Text strings to store in RVF file or stream

```
property DocProperties: TStringList;
```

(introduced in v1.8)

This property is public,so it is not available at design time.

Items of this list can contain line break characters (#13 and #10), but cannot contain #01 and #02 characters.

If *rvfoSaveDocProperties* is in RVFOptions [152], these strings are saved in RVF.

If *rvfoLoadDocProperties* is in RVFOptions [152], methods for loading RVF load these strings. Methods for inserting RVF ignore these strings.

Method TRichView.Clear [175] clears DocProperties.

See also methods for saving RVF (they can save this property):

- SaveRVF [220];
- SaveRVFToStream [220].

See also methods for loading RVF (they can load this property):

- LoadRVF [211];
- LoadRVFFromStream [211].

See also methods of TRichViewEdit for inserting RVF in the caret position (they ignore this property):

- InsertRVFFromFileEd [324];
- InsertRVFFromStreamEd [323];
- PasteRVF [331].

See also methods for inserting RVF (they ignore this property):

- InsertRVFFromStream [206];
- AppendRVFFromStream [173].

See also properties:

- RVFOptions [152].

See also:

- Saving and loading RichView documents [58];
- RVF [59].

1.4.1.1.2.15 TRichView .DocumentHeight

Height of document (height of scrollable area)

```
property DocumentHeight: Integer;
```

Document must be formatted to obtain a correct value.

Example:

```
if MyRichView.DocumentHeight>Screen.Height then
  MyRichView.Height := Screen.Height
else
  MyRichView.Height := MyRichView.DocumentHeight;
```

See also:

- Scrolling 46.

1.4.1.1.2.16 TRichView .DoInPaletteMode

Defines behavior of RichView when drawn in 256-color mode display.

```
type
  TRVPaletteAction =
    (rvpaDoNothing, rvpaAssignPalette,
     rvpaCreateCopies, rvpaCreateCopiesEx);

property DoInPaletteMode: TRVPaletteAction.
```

Note: TRVPaletteAction is declared in RVScroll unit.

Please always set DoInPaletteMode to rvpaDoNothing under Delphi 2 and C++Builder 1.Palette mode improvements are not available for these versions.

This property affects only displaying in-256 color mode and does nothing in highcolor and truecolor modes.

Windows XP does not support 256-color mode.

Mode	Meaning
rvpaDoNothing	RichView does nothing with palette. Use if you have no pictures in RichView, or all pictures are in VGA 16-color palette.
rvpaAssignPalette	When working in 256-color mode, RichView creates palette and assigns it to all pictures (including background). This will affect all pictures that already in RichView, and all pictures that will be inserted later (when in 256-color mode). This is also not very good option, because pictures lose their quality, and it can't be restored when switching to highcolor/truecolor mode. Output to RVF, HTML, clipboard, printer will lose quality of pictures. So use this option if you want only to display document.
rvpaCreateCopies	(default and preferred option) In 256-color mode, RichView creates copies of all pictures (including background) and assign palette to them. This will affect all pictures that already in RichView, and all pictures that will be inserted later (when in 256-color mode). This work is absolutely "transparent" for you, you will work with source pictures that remain unchanged.
rvpaCreateCopiesEx	For future versions, do not use (idea: copies of pictures

	may be created not in format of source picture, but in some better format)

Notes:

1. If you have other controls on form which also change palette (such as TImage, or buttons with multicolor glyphs), you can not achieve required results.
2. If you use multicolor *hotspots* or *bullets* in RichView, it's recommended to have two sets of them: 16color for 16- and 256-color modes, and multicolor for others.
3. RichView can't change palette of bitmaps inside metafiles; do not use TMetafile containing multicolor bitmaps in 256-color mode.
4. RichView can't update information about palette when user switches color resolution without restarting Windows. Please do it yourself, as it's shown below.
5. If you create RichViews, RichViewEdits, RVPrintPreviews, RVPrints at run-time (such as MyRichView := TRichView.Create(Self)), you need to call UpdatePaletteInfo methods of such controls after you have created them.

In the main form write...

In interface section:

```
procedure WMDisplayChange(var Message: TWMDisplayChange); message WM_DISPLAYCHANGE;
```

In implementation:

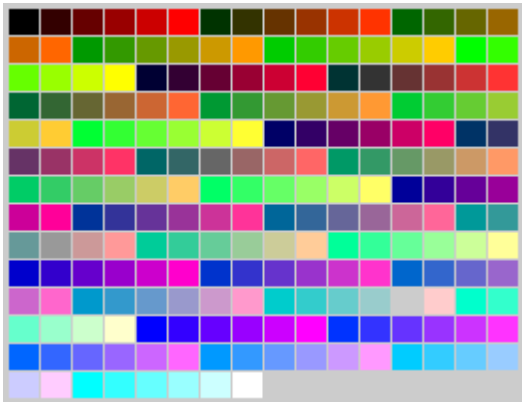
```
procedure TMyForm.WMDisplayChange(var Message: TWMDisplayChange);
begin
  // Call UpdatePaletteInfo methods of all your
  // RichViews, RichViewEdits, and RVPrints (if you use print preview)
  // for example:
  MyRichViewEdit.UpdatePaletteInfo;
  MyRVPrint.UpdatePaletteInfo;
end;
```

Default value:

rvpaCreateCopies

(or *rvpaDoNothing* in Delphi 2 and C++Builder 1)

RichView uses the palette displayed below (the same as Microsoft Internet Explorer uses):



If you need to use your own palette, let me know, I will create a public property for it. Now you can override the protected dynamic method GenerateLogPalette (introduced in TRVScroller).

See also methods:

- UpdatePaletteInfo.

See also methods of TRVPrint:

- UpdatePaletteInfo 518.

1.4.1.1.2.17 TRichView.FirstItemVisible

Index of the first visible (in window) item, or -1 if the document is empty.

```
property FirstItemVisible: Integer;
```

You can use this property (for example, in OnVScrolled 513 event handler) to get the index of the first visible item.

This property does not take horizontal scrolling into account, only vertical one.

Note: returning value of this property requires some calculations; so if you need to use it several times in the same procedure, assign its value to some temporary variable and then use this variable.

See also properties:

- LastItemVisible 144.

See also:

- Vertical scrolling 46.

1.4.1.1.2.18 TRichView.FirstJumpNo

Hypertext **id** for the first hyperlink in the document.

```
property FirstJumpNo: Integer;
```

When you append hypertext items (*hotspots* 97, *hot-pictures* 93 or text of hypertext style), RichView assigns hypertext identifiers to them. The first such item has **id**=FirstJumpNo, the next one – FirstJumpNo+1 and so on.

All hypertext items have hypertext **id**s. **Id**s are incremented according to their order in the document, from top to bottom.

This property is useful if you want to use one OnJump 247 event handler for several RichViews.

Default value:

0

See also:

- RichView hypertext 35.

1.4.1.1.2.19 TRichView.ItemCount

Number of items in RichView

```
property ItemCount: Integer;
```

See also:

- Item types 29;
- Adding items 43;
- Obtaining items 50;
- Modifying items 51.

1.4.1.1.2.20 TRichView.LastItemVisible

Index of the last visible (in window) item, or -1 if the document is empty.

```
property LastItemVisible: Integer;
```

You can use this property (for example, in OnVScrolled 513 event handler) to get the index of the last visible item.

This property does not take horizontal scrolling into account, only vertical one.

Note: returning value of this property requires some calculations; so if you need to use it several times in the same procedure, assign its value to some temporary variable and then use this variable.

See also properties:

- FirstItemVisible 144.

See also:

- Vertical scrolling 46.

1.4.1.1.2.21 TRichView.LeftMargin

Left margin of document

```
property LeftMargin: Integer;
```

This value is measured in "screen" pixels, so when printing it's recalculated according to the printer resolution, and margins on the page look like margins on the screen.

Left margin can be used for selecting lines with the mouse.

In TRichViewEdit 282, you can change this property as an editing operation, see TRichViewEdit.SetIntPropertyEd 337.

See also properties:

- RightMargin 149;
- TopMargin 158;
- BottomMargin 137.

See also indents in paragraph style 448.

See also properties of TRVPrint:

- Margins 486.

See also properties of TRVStyle:

- LineSelectCursor 394.

See also:

Additional information about horizontal scrolling, margins, indents, etc. 46

1.4.1.1.2.22 TRichView.LineCount

Obsolete, use ItemCount 144.

```
property LineCount : Integer;
```

See also

- GetLineNo 198.

1.4.1.1.2.23 TRichView.MaxLength

Allows to to limit a number of characters per line.

```
property MaxLength: Integer;
```

(introduced in version 10)

Set this property to positive value to limit a number of characters per line. Non-text items are treated like one character.

This property affects only word wrapping.

This property is ignored if WordWrap 159 = *False*. This property is ignored for paragraphs having *rvpaoNoWrap* in Options 453.

Default value:

0

1.4.1.1.2.24 TRichView .MaxTextWidth

If set to positive value, it is a maximal width for text wrapping (text wraps when its length in pixels exceeds this value)

```
property MaxTextWidth: Integer;
```

This property does not affect printing.

In TRichViewEdit 282, you can change this property as an editing operation, see TRichViewEdit.SetIntPropertyEd 337.

Default value:

0

See horizontal scrolling 46 for details.

See also properties:

- MaxLength 145.

1.4.1.1.2.25 TRichView .MinTextWidth

Minimum width for text wrapping. This value allows to increase width of scrollable area in TRichView.

```
property MinTextWidth : Integer;
```

This property does not affect printing.

In TRichViewEdit 282, you can change this property as an editing operation, see TRichViewEdit.SetIntPropertyEd 337.

Default value:

0

See horizontal scrolling 46 for details.

1.4.1.1.2.26 TRichView .NoteText

Text for references to the parent note 108.

```
property NoteText: String;
```

(introduced in version 10)

Normally, references 108 must be placed in Document 586 for footnotes 104, endnotes 104 or sidenotes 105. In this case, they display the same text as their parent note.

References 108 should not appear in TCustomRichView, but the user can copy them here. In this case, they display text specified in this property.

If this TCustomRichView is used for displaying/editing Document 586 for some note, you should set value of this property equal to NoteText 587 of this note.

Unicode note: in Delphi/C++Builder 2009 or newer, this property contains Unicode text. For older versions, it contains ANSI text.

Default value:

'?'

1.4.1.1.2.27 TRichView .Options

Options

```
type
  TRVOption = (rvoAllowSelection, rvoSingleClick, rvoScrollToEnd,
    rvoClientTextWidth,
    rvoShowCheckpoints, rvoShowPageBreaks,
```

```
    rvoShowSpecialCharacters, rvoShowHiddenText,
    rvoTagsArePChars,
    rvoAutoCopyText,  rvoAutoCopyUnicodeText
    rvoAutoCopyRVF, rvoAutoCopyImage,
    rvoAutoCopyRTF,
    rvoFormatInvalidate,
    rvoDblClickSelectsWord, rvoRClickDeselects,
    rvoDisallowDrag, rvoShowItemHints, rvoShowGridLines,
    rvoFastFormatting, rvoPercentEncodedURL);
  TRVOptions = set of TRVOption;

property Options: TRVOptions;
```

Note: TRVOption and TRVOptions are declared in RVScroll unit.

Default value

For Delphi/C++Builder 2009 or newer: [*rvoAllowSelection, rvoScrollToEnd, rvoAutoCopyUnicodeText, rvoAutoCopyImage, rvoAutoCopyRVF, rvoAutoCopyRTF, rvoDblClickSelectsWord, rvoRClickDeselects, rvoFormatInvalidate, rvoShowPageBreaks, rvoShowGridLines, rvoFastFormatting*]
For older versions: the same, but with *rvoAutoCopyText* instead of *rvoAutoCopyUnicodeText*.

General Options

Option	Meaning
rvoTagsArePChars	Obsolete, does nothing.
rvoClientTextWidth	If set, text always wraps to fit client width of RichView. Also affects breaks 93 and tables 98. See details 46. Can be changed: any time, but you need to reformat 214 and repaint RichView after changing this option, if the document was already formatted.
rvoScrollToEnd	If set (default), RichView scrolls to the end of document when FormatTail 183 method is called. If cleared, document does not scroll in this case Can be changed: any time.
rvoSingleClick	If set, OnRVDblClick 250 event is generated on single mouse click If cleared (default), it is generated on double mouse click Can be changed: any time.
rvoFormatInvalidate	If set (default), Format 182 and FormatTail 183 methods repaint component. Can be changed: any time.
rvoDisallowDrag	If set, dragging 55 from this TRichView is not allowed. Can be changed: any time.
rvoFastFormatting	Increase performance in WinNT-based OS at the cost of some resources
rvoPercentEncodedURL	If set, it is assumed that URLs (stored in tags or provided in OnWriteHyperlink 267 event) are %-encoded, so no additional encoding/decoding is necessary when working with RTF, HTML, and DocX. By default (if not set), the component %-encodes hyperlink targets saved to RTF, HTML, DocX, and %-decodes when loading RTF.

Selection Options

Can be changed: any time.

Option	Meaning
rvoAllowSelection	If set (default), user can select content of RichView using mouse or keyboard. Please do not clear this option for TRichViewEditor! See: working with selection 47.
rvoDblClickSelectsWord	If set (default), double click on word selects it (and double click on non-text item selects this item)
rvoRClickDeselects	If set (default), right click outside the selected area deselects (and moves caret to the clicked position in TRichViewEdit 282)

Show/Hide Options

Option	Meaning
rvoShowCheckpoints	If set, all *checkpoints* are shown (as dotted horizontal lines, or by OnDrawCheckpoint 411). See details 30. Can be changed: any time, you need to repaint RichView after changing this option
rvoShowPageBreaks	If set (default), page breaks are shown as horizontal lines. Can be changed: any time, you need to repaint RichView after changing this option
rvoShowSpecialCharacters	If set, nonprinting characters are displayed as special marks: spaces, nonbreaking spaces, paragraph breaks, line breaks, tabs 89, soft hyphens in Unicode text, lines to left- and right-aligned items, places of insertion of text boxes 107. Can be changed: any time, you need to reformat 214 RichView after changing this option, if the document is already formatted. See also: RVVisibleSpecialCharacters 665, RichViewShowGhostSpaces 656 typed constants.
rvoShowHiddenText	If set, hidden items 42 are displayed. Can be changed: any time, you need to reformat 214 RichView after changing this option, if the document is already formatted.
rvoShowItemHints	If set, items' hints are shown. Popup hints are shown only if ShowHints property is set to True. See item properties 630 and OnItemHint 246 event.
rvoShowGridLines	If set, table 98 grid lines 392 are shown.

Clipboard Options (See Details 48)

If set , RichView copies the selection to the Clipboard in the specified formats when the user presses Ctrl + Insert or Ctrl + C. These options also affect CopyDef 177. These options can be changed at any time.

Option	Format to Copy
rvoAutoCopyText	ANSI text
rvoAutoCopyUnicodeText	Unicode text

rvoAutoCopyRVF	RVF 59 (RichView Format)
rvoAutoCopyRTF	RTF (Rich Text Format)
rvoAutoCopyImage	Bitmap or metafile (if the image is selected)

1.4.1.1.2.28 TRichView.PageBreaksBeforeItems

Specifies if the **Index**-th item must start a new page when printing

```
property PageBreaksBeforeItems[Index: Integer]: Boolean
```

(Introduced in version 1.2)

Set PageBreaksBeforeItems[ItemNo] to *True* if you want the ItemNo-th item to start a new page. This assignment forces the item to start a new paragraph.

If PageBreaksBeforeItems[ItemNo]=*True*, a text flow for this item is cleared for both left and right side, i.e. such item is placed as if ClearLeft 138[ItemNo] and ClearRight 138[ItemNo] = *True*. Because of this, assignment to this property requires reformatting.

RichView never prints blank pages. So, for example, PageBreaksBeforeItems[0] has no effect (because it will start a new page in any case).

Editor has additional methods for working with explicit page breaks.

See also:

- Options 146 (*rvoShowPageBreaks*).

See also methods:

- AssignSoftPageBreaks 174.

See also properties and events of TRVStyle:

- PageBreakColor 398;
- OnDrawPageBreak 413.

See also methods of TRichViewEdit:

- InsertPageBreak 321;
- RemoveCurrentPageBreak 334.

1.4.1.1.2.29 TRichView.RightMargin

Right margin of document

```
property RightMargin: Integer;
```

This value is measured in "screen" pixels, so when printing it's recalculated according to the printer resolution, and margins on pages look like margins on the screen.

In TRichViewEdit 282, you can change this property as an editing operation, see TRichViewEdit.SetIntPropertyEd 337.

See also properties:

- LeftMargin 145;
- TopMargin 158;
- BottomMargin 137.

See also indents in paragraph style 448

See also properties of TRVPrint:

- Margins 486.

See also properties of TRVStyle:

- LineSelectCursor 394.

See also:

Additional information about horizontal scrolling, margins, indents, etc. 46

1.4.1.1.2.30 TRichView .RTFOptions

Options for saving RTF (Rich Text Format). Some options are used for saving DocX files as well.

```
type
  TRVRTFOption = (rvrtfSaveStyleSheet,
    rvrtfDuplicateUnicode, rvrtfSaveEMFAsWMF,
    rvrtfSaveJpegAsJpeg, rvrtfSavePngAsPng,
    rvrtfSaveBitmapDefault,
    rvrtfSaveEMFDefault, rvrtfSavePicturesBinary,
    rvrtfPNGInsteadOfBitmap,
    rvrtfSaveDocParameters, rvrtfSaveHeaderFooter);
  TRVRTFOptions = set of TRVRTFOption;

property RTFOptions: TRVRTFOptions;
```

(introduced in versions 1.3, 10, 11)

Note: TRVRTFOption and TRVRTFOptions are declared in RVStyle unit.

Options for RTF

Option	Meaning
rvrtfSaveStyleSheet	Used only if UseStyleTemplates 158=*False*. If set (not recommended), RichView saves a style sheet in RTF; this style sheet is created from text and paragraph styles having Standard 431 =*True*. If UseStyleTemplates 158=*True*, the component always saves Style 156.StyleTemplates 402 as a style sheet.
rvrtfDuplicateUnicode	If set, RichView saves an ANSI equivalent of Unicode text 64. A conversion is based on Charset 434 property of text styles. Turning off this option increases saving speed and produces smaller RTF file. Unicode-enabled RTF readers just ignore ANSI equivalent of Unicode characters. However, ANSI text is necessary for RTF readers that cannot read Unicode.
rvrtfSaveEMFAsWMF	If set, RichView saves EMF (enhanced (Win32) metafiles) as WMF (Windows (Win3.1) metafiles). Some RTF readers do not understand EMF in RTF.
rvrtfSaveJpegAsJpeg	If set, RichView saves TJpegImage as JPEG. Otherwise, RichView saves JPEG images in other formats (see below). Some RTF readers do not understand JPEG images in RTF.
rvrtfSavePngAsPng	If set, RichView saves PNG images (PNG class is registered with RVGraphicHandler 662.RegisterPngGraphic) as PNG. Otherwise, RichView saves PNG images in other formats (see below). Some RTF readers do not understand PNG images in RTF.
rvrtfSaveBitmapDefault	If set, RichView saves graphics formats other than bitmaps, metafiles (and optionally Jpegs) as bitmaps. If not set, as WMF or EMF (see below). See also *rvrtfPNGInsteadOfBitmap* below. (unfortunately, many RTF reader do not understand WMF with mapping mode = MM_TEXT, which RichView uses for saving other formats; some other RTF readers do not understand bitmaps).
rvrtfSaveEMFDefault	If set (and rvrtfSaveBitmapDefault is not set), RichView saves graphics formats other than bitmaps (and optionally Jpegs) as EMF. If not set, as WMF

rvrtfSavePicturesBinary	If set, pictures are saved in binary format. Less in size (twice), but such RTF becomes a binary file (and cannot be stored in MEMO field of data bases)
rvrtfPNGInsteadOfBitmap	If included, and PNG class is registered with RVGraphicHandler[662] .RegisterPngGraphic, bitmaps are saved as PNG (including saving of non-RTF picture types, if *rvrtfSaveBitmapDefault* is included). Not all RTF readers understand PNG.

Options for RTF and DocX

Option	Meaning
rvrtfSaveDocParameters	If included, properties of DocParameters[140] are saved in RTF and DocX.
rvrtfSaveHeaderFooter	If included, documents assigned by RTFReadProperties[151] .SetHeader[276] and .SetFooter[276] are saved as a header and footer in RTF and DocX.

Default value

[*rvrtfDuplicateUnicode*, *rvrtfSaveEMFAsWMF*, *rvrtfSaveJpegAsJpeg, rvrtfSavePngAsPng*]

See also methods:

- SaveRTF[219];
- SaveRTFToStream[220];
- CopyRTF[178].

See also:

- Saving and loading in RichView[58];
- Pictures[90].

1.4.1.1.2.31 TRichView.RTFReadProperties

A set of properties controlling RTF (Rich Text Format) import.

```
property RTFReadProperties: TRVRTFReaderProperties[268];
```

Read the help on TRVRTFReaderProperties for details.

Subproperties of this class affect the following methods:

- TRichView.LoadRTF[209];
- TRichView.LoadRTFFromStream[210];
- TRichViewEdit.PasteRTF[330] (and Paste[328]);
- TRichViewEdit.InsertRTFFromStreamEd[323];
- TRichViewEdit.InsertRTFFromFileEd[324];
- drag&drop[55] when dropping RTF file.

See also:

- RTFOptions[150] (options for RTF export).

1.4.1.1.2.32 TRichView.RVData

This read-only property points to an object containing RichView document.

Each TRichView control owns one such object.

Similar objects represent cells[567] in tables[528].

Many RichView methods actually call methods of this object inside, for example:

```
procedure TRichView.AddNL(const s: String; StyleNo, ParaNo: Integer);
begin
```

```
  RVData.AddNL(s, StyleNo, ParaNo);
end;
```

A class of RVData is derived from the abstract base class TCustomRVFormattedData[608], which is in its order derived from TCustomRVData.

This object can be used for creating procedures working with any document object (main TRichView or cells).

For example, see AllUpperCase procedure in Controls, Documents, Items[72].

1.4.1.1.2.33 TRichView .RVFOptions

Options for reading and saving RVF (RichView Format)

```
type
  TRVFOption =
    (rvfoSavePicturesBody, rvfoSaveControlsBody,
    rvfoIgnoreUnknownPicFmt, rvfoIgnoreUnknownCtrls,
    rvfoIgnoreUnknownCtrlProperties
    rvfoConvUnknownStylesToZero,
    rvfoConvLargeImageIdxToZero,
    rvfoSaveBinary,
    rvfoUseStyleNames,
    rvfoSaveBack, rvfoLoadBack,
    rvfoSaveTextStyles, rvfoSaveParaStyles,
    rvfoSaveLayout, rvfoLoadLayout,
    rvfoSaveDocProperties, rvfoLoadDocProperties,
    rvfoCanChangeUnits, rvfoSaveStyleTemplatesOnlyNames);
  TRVFOptions = set of TRVFOption;

property RVFOptions: TRVFOptions;
```

Note: TRVFOption and TRVFOptions are defined in RVStyle unit.

Options for Saving

Option	Meaning
rvfoSavePicturesBody	▪ If set, full information about all pictures in RichView are saved in RVF. ▪ If not set, OnRVFPictureNeeded[252] event occurs on loading.
rvfoSaveControlsBody	▪ If set, full information about all controls in RichView are saved in RVF. ▪ If not set, OnRVFControlNeeded[251] event occurs loading.
rvfoSaveBinary	▪ If set, all non-text data are saved in RVF as they are (pictures, controls, tables, background, styles, Unicode text). ▪ If not set, all binary data are converted to text (hexadecimal string); RVF becomes a plain text format, like RTF, but file size is increased.
rvfoSaveBack	*(since version 1.2)* If set, background (color and image) is saved in RVF. This flag affects only saving the whole document; selection is never saved (or copied to the Clipboard) with background. (see *rvfoLoadBack* below)
rvfoUseStyleNames *(almost obsolete)*	If set, TRichView saves quoted names of text and paragraph styles instead of their indices **Pluses:** ▪ content of file remains synchronized with collections of styles even

	if you delete or insert new styles; **Minuses:** ▪ increased file size; ▪ names of styles must be unique; ▪ files become sensitive to changing style names. It's not recommended to use this options together with *rvfoSaveTextStyles* and *rvfoSaveParaStyles*.
rvfoSaveTextStyles	If set, collection of text styles associated with this RichView (TextStyles 404 property of the linked 156 TRVStyle component) is saved in RVF. See RVFTextStylesReadMode 154 property. This option affects saving style templates.*
rvfoSaveParaStyles	If set, collections of paragraph and list styles associated with this RichView (ParaStyles 398 and ListStyles 396 properties of the linked 156 TRVStyle component) are saved in RVF. See RVFParaStylesReadMode 154 property. This option affects saving style templates.*
rvfoSaveLayout	If set, the following properties are stored in RVF: Left 145-, Right 149-, Top 158-, BottomMargin 137, MinTextWidth 146, MaxTextWidth 146, BiDiMode 137. Layout is saved only when saving the whole document (not selection). (see *rvfoLoadLayout* below)
rvfoSaveDocProperties	If set, DocProperties 141 and DocParameters 140 are saved in RVF. (see *rvfoLoadDocProperties* below)
rvfoSaveStyleTemplatesOnlyNames	If set, when style templates must be saved in RVF*, only their names are saved. When loading such files, existing style templates are applied to loaded text and paragraph styles. This option is useful if you want to create multiple documents using the same collection of style templates. Since style templates are not saved in RVF in this mode, an application must save 429 them itself elsewhere.

* If the both *rvfoSaveTextStyles* and *rvfoSaveParaStyles* are included, and UseStyleTemplates 158=*True*, Style 156.StyleTemplates 402 are saved as well.

Options for Loading

Option	Meaning
rvfoLoadBack	If not set, RichView ignores background information, saved in RVF. (see *rvfoSaveBack* above)
rvfoIgnoreUnknownPicFmt	▪ If set, RichView skips pictures of unknown (unregistered) classes. ▪ if not set, RVF reading methods return False (failure value) if pictures of unknown classes exist in RVF
rvfoIgnoreUnknownCtrls	The same as *rvfoIgnoreUnknownPicFmt*, but for controls
rvfoIgnoreUnknownCtrlProperties	If set, RichView ignores errors occurred while reading inserted controls
rvfoConvLargeImageIdxToZero	▪ If set, too large (>=Imagelist.Count) image indices for *bullets* and *hotspots* are converted to 0. ▪ if not set, RVF reading methods return False (failure value) in such case.
rvfoConvUnknownStylesToZero	▪ If set, RichView converts too large (>= RichView.Style.TextStyles 404.Count, or RichView.Style.ParaStyles 398.Count) style indices to

	0. ▪ If not set, RVF reading methods return False (failure value) in such case.
rvfoLoadLayout	If not set, RichView ignores layout information saved in RVF (see *rvfoSaveLayout* above)
rvfoLoadDocProperties	If set, DocProperties[141] and DocParameters[140] are loaded from RVF (but not by methods for RVF insertion) (see *rvfoSaveDocProperties* above)
rvfoCanChangeUnits	If set, the component can change Style[385].Units[404] to the value read from RVF. This change is possible only when RVF is loaded, not inserted.

See RVF Overview[59] for details.

Default value:

[*rvfoSavePicturesBody, rvfoSaveControlsBody, rvfoSaveBinary, rvfoSaveDocProperties, rvfoLoadDocProperties*];

Default value for components, placed on form at design time (can be changed in TRichView component editor)[134]:

[*rvfoSavePicturesBody, rvfoSaveControlsBody, rvfoSaveBinary, rvfoSaveDocProperties, rvfoLoadDocProperties, rvfoSaveTextStyles, rvfoSaveParaStyles*];

See also:

- RVFWarnings[155]

1.4.1.1.2.34 TRichView.RVFParaStylesReadMode

Specifies how paragraph styles[398] and list styles[396] are loaded from RVF files or streams.

```
property RVFParaStylesReadMode: TRVFReaderStyleMode[633];
```

(Introduced in version 1.5)

See topic on TRVFReaderStyleMode[633] type for details.

Paragraph styles are saved in RVF optionally.

Default value:

rvf_sInsertMerge

See properties:

- RVFTextStylesReadMode[154];
- RVFOptions[152].

See also:

- RVF Overview[59]

1.4.1.1.2.35 TRichView.RVFTextStylesReadMode

Specifies how text styles[404] are loaded from RVF files or streams.

```
property RVFTextStylesReadMode: TRVFReaderStyleMode[633];
```

(Introduced in version 1.5)

See topic on TRVFReaderStyleMode[633] type for details.

Text styles are saved in RVF optionally.

Default value:

rvf_sInsertMerge

See properties:

- RVFParaStylesReadMode[154];

- RVFOptions 152.

See also:

- RVF Overview 59.

1.4.1.1.2.36 TRichView.RVFWarnings

Information about the last performed RVF reading operation.

```
type
  TRVFWarning = (
    rvfwUnknownPicFmt, rvfwUnknownCtrls,
    rvfwConvUnknownStyles, rvfwConvLargeImageIdx,
    rvfwConvToUnicode, rvfwConvFromUnicode,
    rvfwInvalidPicture, rvfwUnknownCtrlProperties, rvfwConvUnits);
  TRVFWarnings = set of TRVFWarning;

property RVFWarnings: TRVFWarnings;
```

Note: TRVFWarning and TRVFWarnings are defined in RVStyle unit.

You can use this property to know about warnings or errors that may be occurred during the last reading from RVF.

Depending on RichView.RVFOptions 152 some cases could be considered as warnings (and reading method was successfully finished) or errors.

Warning	Meaning
rvfwUnknownPicFmt	Pictures of unknown classes exist in RVF. See *rvfoIgnoreUnknownPicFmt.*
rvfwUnknownCtrls	Controls of unknown classes exist in RVF. See *rvfoIgnoreUnknownCtrls.*
rvfwConvLargeImageIdx	*Bullets* or *hotspots* with too large image indices exist in RVF. See *rvfoConvLargeImageIdxToZero.*
rvfwConvUnknownStyles	Text, paragraph or list styles with too large indices exist in RVF. See *rvfoConvUnknownStylesToZero.*
rvfwConvToUnicode *rvfwConvFromUnicode*	There was a conversion from ANSI text in RVF file to Unicode in RichView or vice versa. This is a result of loading RVF file with set of styles different from one which was used when saving. A conversion is performed automatically, without generating errors.
rvfwInvalidPicture	Some pictures were replaced with InvalidPicture 393.
rvfwUnknownCtrlProperties	There was error when reading inserted controls (for example, unknown properties)
rvfwConvUnits	RVF file has different units than Style 385.Units 404, and a conversion of units was made. This warning is not set if units were changed because of *rvfoCanChangeUnits* included in RVFOptions 152. A conversion is performed automatically, without generating errors.

See RVF Overview 59 for details.

Methods reading RVF:

- LoadRVFFromStream 211;

- InsertRVFFromStream 206;
- AppendRVFFromStream 173;
- LoadRVF 211.

Methods of RichViewEdit reading RVF:

- InsertRVFFromFileEd 324;
- InsertRVFFromStreamEd 325;

(also PasteRVF 331 and Paste 328, but in this version you can't detect when Clipboard reading operation is finished).

1.4.1.1.2.37 TRichView.SelectionHandlesVisible

Shows or hides selection handles for using on touch screens.

```
property SelectionHandlesVisible: Boolean;
```

(introduced in version 15)

This property exists when the component is compiled in Delphi 2010 or newer.

By default, selection handles are shown if the selection was created by a double or a triple tap (double tap selects a word, triple tap selects a paragraph). They are hidden automatically when the selection becomes empty.

You can use this property to show or hide handles manually. Assigning *True* to this property can show handles even if a selection is empty.

For example, if you want to show handles even on a single tap, assign SelectionHandlesVisible=True in OnRVMouseUp 254, if ssTouch is included in the Shift parameter of this event.

Default value:

True

See also properties of TRVStyle:

- SelectionHandleKind 398

1.4.1.1.2.38 TRichView.SingleClick

```
property SingleClick : Boolean;
```

Obsolete, please use *rvoSingleClick* in Options 146;

1.4.1.1.2.39 TRichView.Style

Style of TRichView: 

```
property Style: TRVStyle 385;
```

This property provides link to TRVStyle component.

TRVStyle define many properties for document appearance, and, the most important, it contains collections of text, paragraph and list styles for TRichView document.

RichView can display its contents only if its style is defined.

1.4.1.1.2.40 TRichView.StyleTemplateInsertMode

Controls how Style 156.StyleTemplates 402 affect insertion of RTF and RVF files or streams.

```
type
  TRVStyleTemplateInsertMode =
    (rvstimUseTargetFormatting, rvstimUseSourceFormatting, rvstimIgnoreSourceStyleTemplates
```

```
property StyleTemplateInsertMode: TRVStyleTemplateInsertMode;
```

(introduced in version 14)

This property is used only if UseStyleTemplates[158]=*True*.

Mode	Meaning
rvstimUseTargetFormatting	Use formatting of the current (target) document
rvstimUseSourceFormatting	Use formatting of the inserted (source) document
rvstimIgnoreSourceStyleTemplates	Ignore styles in the inserted (source) document.

1) rvstimUseTargetFormatting

If the source document and the target document have styles templates of the same name, text (and paragraphs) of this style template is formatted according to the target style template.

For example, let the source document has a style template named "heading 1" defining text color=red, and some text formatted as "heading 1"+bold, i.e. it looks red and bold. Let the target document has a style template of the same name, defining text color=black. When this text is inserted, it is formatted as "heading 1"+bold, i.e. it looks black+bold.

Only style templates having new names are added from the source document to the target document.

2) rvstimUseSourceFormatting

If the source document and the target document have style templates of the same name, text (and paragraphs) of this style template is formatted according to the source style template.

In the example above, this text is inserted as "heading 1"+red+bold, so it looks like in the source document, despite of linking to the target document's "heading 1" instead of the source document's "heading 1".

Only style templates having new names are added from the source document to the target document.

3) rvstimIgnoreSourceStyleTemplates

Style templates from the source document are ignored. The inserted paragraphs are linked to the target document's "Normal" style template[428] (if it exists). The inserted text is unstyled, except for hyperlinks[445] (they are linked to the target document's "Hyperlink" style, if it exists).

In the example above, this text is inserted as "Normal"+red+bold, so it looks like in the original document.

Note

When inserting unstyled paragraphs, the "Normal" style template is applied to them. To insert them unstyled in modes (1) and (2), you can assign the global variable RichViewAutoAssignNormalStyleTemplate[653] = *False*. In the mode (3), "Normal" is always applied.

Default value

rvstimUseTargetFormatting

See also:

- Styles and style templates[40].

1.4.1.1.2.41 TRichView.TabNavigation

Mode of Tab and Shift+Tab navigation

```
type
  TRVTabNavigationType =
    (rvtnNone, rvtnTab, rvtnCtrlTab)

property TabNavigation: TRVTabNavigationType;
```

(introduced in version 1.3)

Mode	Meaning
rvtnNone	No special actions for Tab (default for TRichViewEdit). A focus (dotted) rectangle is not drawn around hyperlinks, it's impossible to open hyperlink with Enter.
rvtnTab	Tab-navigation through links and controls using Tab and Shift+Tab (default for TRichView);
rvtnCtrlTab	Tab navigation through links using Ctrl+Tab and Ctrl+Shift+Tab. Has a limited application since users still need to use Tab and Shift+Tab when the focus is set on inserted control.

Note: a special tab navigation is not supported in editor. Although it has TabNavigation property, it's impossible to change its value from rvtnNone.

See also:

- GetFocusedItem 188 method.

1.4.1.1.2.42 TRichView .TopMargin

Top margin of document

```
property TopMargin: Integer;
```

This value is measured in "screen" pixels, so when printing it's recalculated according to the printer resolution, and margins on pages look like margins on the screen.

In TRichViewEdit 282, you can change this property as an editing operation, see TRichViewEdit.SetIntPropertyEd 337.

See also properties:

- LeftMargin 145;
- RightMargin 149;
- BottomMargin 137.

See also properties of TRVPrint:

- Margins 486.

See also:

Additional information about horizontal scrolling, margins, indents, etc. 46

1.4.1.1.2.43 TRichView .UseStyleTemplates

Allows or disallows using Style 156.StyleTemplates 402.

```
property UseStyleTemplates: Boolean;
```

(introduced in version 14)

If UseStyleTemplates=*True*:

- StyleTemplates are saved in RVF (also requires both *rvfoSaveTextStyles* and *rvfoSaveParaStyles* included in RVFOptions 152);
- StyleTemplates are loaded from RVF (also requires RVFTextStylesReadMode 154=RVFParaStylesReadMode 154=*rvf_sInsertMerge*);
- StyleTemplates are saved in RTF and DocX (as a style sheet);
- StyleTemplates are loaded from RTF (also requires RTFReadProperties 151.TextStyleMode 274 =RTFReadProperties.ParaStyleMode 272=*rvrsAddIfNeeded*) ;
- RTF and RVF files/streams are inserted according to StyleTemplateInsertMode 156;
- "Normal" style template (if it exists) is assigned to unstyled paragraphs when loading RTF and RVF files/ streams (see RichViewAutoAssignNormalStyleTemplate 653);

- the component provides that TextStyle.ParaStyleTemplateId[439]=ParaStyle.StyleTemplateId[432] for all text items[88] and non-text items linked to a text style (tabs[89], labels[101], sequences[102], footnotes[104], endnotes[103], note references[108]); if they are not equal, the component assigns another text style to this item, with the correct ParaStyleTemplateId, adding this text style to Style[385].TextStyles[404] if necessary.
- in editor, when Enter is pressed, the next paragraph is formatted according to NextId[462] property of the current paragraph's style template

Default value

False

See also:

- Styles and style templates[40].

1.4.1.1.2.44 TRichView.VAlign

Vertical align of the document relative to the control window: top, center, or bottom.

```
property VAlign: TTextLayout;
```

(introduced in v1.9)

This property is in effect if height of the document is less than height of the TRichView window.

See also:

- ClientToDocument[176] method.

1.4.1.1.2.45 TRichView.VSmallStep

Number of pixels in one step of vertical scrollbar

```
property VSmallStep: Integer;
```

When VScrollPos[512] is changes by 1, the window is scrolled by VSmallStep pixels.

This is one "RVSU", see Scrolling in RichView[46]

Default value:

10

See also properties:

- VScrollPos[512];
- VScrollMax[512];
- VScrollVisible[512].

1.4.1.1.2.46 TRichView.WordWrap

Allows to disable word wrapping.

```
property WordWrap: Boolean;
```

(introduced in version 10)

If *True*, documents are formatted normally.

If *False*, all word wrapping is disabled. Unlike *rvpaoNoWrap* option[453] of paragraph style, this property should be considered as a control (view) property, not as a document property. It is not stored in RVF[59].

Default value:

True

1.4.1.1.3 Methods

1.4.1.1.3.1 TRichView .AddBreakExTag and Others

These methods add *break* [93] (horizontal line) to the end of document.

```
procedure AddBreakExTag(Width: TRVStyleLength[644]; Style: TRVBreakStyle[624];
  Color: TColor; const Tag: TRVTag[645]);

procedure AddBreak;
procedure AddBreakEx(Width: TRVStyleLength[644]; Style: TRVBreakStyle[624];
  Color: TColor);
procedure AddBreakTag(const Tag: TRVTag[645]);
```

AddBreak, **AddBreakEx**, **AddBreakTag** provide subsets of functionality of AddBreakExTag method:

- AddBreak is equivalent to AddBreakExTag(1, *rvbsLine*, *clNone*, '').
- AddBreakEx(...) is equivalent to AddBreakExTag(..., '').
- AddBreakTag(Tag) is equivalent to AddBreakExTag(1, *rvbsLine*, *clNone*, Tag).

Parameters:

Width – line width (or rectangle height). This value is measured in Style[156].Units[404].

Style – break type;

Color – color of line. If it is equal to *clNone*, the *break* has the color of the 0th text style[404].

Tag – *tag*[34] of this *break*.

For default HTML <hr> appearance, parameters must be: **Style**=*rvbs3d*, **Width**=2.

Methods type: viewer-style.

See also methods:

- Format[182];
- FormatTail[183].

See also methods of TRichViewEdit:

- InsertBreak[317].

See also:

- Item types[29];
- Other methods for appending items[43];
- "Tags"[34].

1.4.1.1.3.2 TRichView .AddBulletExTag and Others

These methods add *bullet*[96] (image from TImageList) to the end of document.

```
procedure AddBulletExTag(const Name: TRVAnsiString[623];
  ImageIndex: Integer; ImageList: TCustomImageList;
  ParaNo: Integer; const Tag: TRVTag[645]);

procedure AddBulletEx(const Name: TRVAnsiString[623]; ImageIndex: Integer;
  ImageList: TCustomImageList; ParaNo: Integer);
procedure AddBullet(ImageIndex: Integer; ImageList: TCustomImageList;
  fromnewline: Boolean); deprecated;
```

AddBulletEx provides a subset of functionality of **AddBulletExTag**:

- AddBulletEx(...) is equivalent to AddBulletExTag(..., '').

AddBullet is deprecated and maintained for backward compatibility (with TRichView versions prior to 1.0):

- AddBullet(ImageIndex, ImageList, *True*) is equivalent to AddBulletEx('', ImageIndex, ImageList, 0).

- AddBullet(ImageIndex, ImageList, *False*) is equivalent to AddBulletEx('', ImageIndex, ImageList, -1).

Alternatives:

- If you want to add a hyperlink-picture from image list, add a *hotspot* instead.
- If you want to add a TGraphic picture, add a picture instead.
- If you want to add a TGraphic picture-hyperlink, add a *hot picture* instead.

Parameters:

Name – name of *bullet*, any string. **Name** must not contain CR and LF characters. RichView does not use Names itself, they are for your own use. This is an ANSI string, regardless of Unicode mode of text styles.

ImageIndex – index of image in **ImageList**.

ImageList – image list for this *bullet*. RichView does not own, does not copy and does not destroy imagelists, it just holds pointers to them. So this image list is not destroyed when the document is cleared.

If **ParaNo**=-1, the methods add the item to the end of the last paragraph. If **ParaNo**>=0, this item starts a new paragraph with the **ParaNo**-th style. (**ParaNo** is an index in the ParaStyles collection of the linked RVStyle component). It defines attributes for the new paragraph.

Tag – *tag* of this *bullet*.

Methods type: viewer-style.

Unicode note: names of non-text items are ANSI strings.

See also methods:

- Format;
- FormatTail;
- SetAddParagraphMode.

See also methods of TRichViewEdit:

- InsertBullet.

See also properties of TRVStyle:

- ParaStyles.

See also:

- Item types;
- Other methods for appending items;
- "Tags".

1.4.1.1.3.3 TRichView.AddCenterLine, AddFromNewLine

These methods add one text item using the **StyleNo**-th text style to the end of document.

```
procedure AddCenterLine(const s: String; StyleNo:Integer); deprecated;
procedure AddFromNewLine(const s: String; StyleNo:Integer); deprecated;
```

AddFromNewLine and AddCenterLine are deprecated and maintained for backward compatibility (with TRichView versions prior to 1.0):

- AddCenterLine(s, StyleNo) is equivalent to AddNL(s, StyleNo, 1).
- AddFromNewLine(s, StyleNo) is equivalent to AddNL(s, StyleNo, 0).

Parameters:

s is a text string to add. It must not contain CR, LF, TAB, FF characters (#13, #10, #9, #12). To add several lines of text use AddTextNL.

StyleNo is an index in the TextStyles collection of the linked RVStyle component, or *rvsDefStyle* constant. It defines font attributes for the text.

Methods type: viewer-style

See also methods:

- AddNL 167.

1.4.1.1.3.4 TRichView .AddControlExTag and Others

These methods add Delphi control 94 to the end of document.

```
procedure AddControlExTag(const Name: TRVAnsiString 623; ctrl: TControl;
  ParaNo: Integer; VAlign: TRVVAlign 648; const Tag: TRVTag 645);

procedure AddControlEx(const Name: TRVAnsiString 623; ctrl: TControl;
  ParaNo: Integer; VAlign: TRVVAlign 648);
procedure AddControl(ctrl: TControl; center: Boolean); deprecated;
```

AddControlEx provides a subset of functionality of **AddControlExTag**:

- AddControlEx(...) is equivalent to AddControlExTag(..., '')

AddControl is deprecated and maintained for backward compatibility (with TRichView versions prior to 1.0):

- AddControl(ctrl, True) is equivalent to AddControlEx('', ctrl, 1, rvwaBaseline);
- AddControl(ctrl, False) is equivalent to AddControlEx('', ctrl, 0, rvwaBaseline).

Parameters:

Name – name of this control item, any string. **Name** must not contain CR and LF characters. RichView does not use Names itself, they are for your own use. This is an ANSI string, regardless of Unicode mode of text styles. Do not confuse with **ctrl**.Name property.

ctrl – control to insert.

VAlign – vertical align of this control, relative to its line, see TRVVAlign 648 for possible values.

If **ParaNo**=-1, the methods add the item to the end of the last paragraph. If **ParaNo**>=0, this item starts a new paragraph with the **ParaNo**-th style. (**ParaNo** is an index in the ParaStyles 398 collection of the linked 156 RVStyle component). It defines attributes for the new paragraph.

Tag – *tag* 34 of this control item. Do not confuse with **ctrl**.Tag property.

Methods type: viewer-style.

Unicode note: names of non-text items are ANSI strings.

See also methods:

- Format 182;
- FormatTail 183;
- SetAddParagraphMode 225.

See also methods of TRichViewEdit:

- InsertControl 318.

See also properties of TRVStyle:

- ParaStyles 398.

See also:

- Item types 29;
- Other methods for appending items 43;
- "Tags" 34.

1.4.1.1.3.5 TRichView.AddFmt

Adds a formatted string assembled from a Pascal format string and an array arguments, using the **StyleNo**-th text style and the **ParaNo**-th paragraph style to the end of document.

```
procedure AddFmt(const FormatStr: String;
  const Args: array of const; StyleNo, ParaNo: Integer);
```

AddFmt(FormatStr, Args, StyleNo, ParaNo) is equivalent to AddNL 167(Format(FormatStr,Args), StyleNo, ParaNo).
(where Format is not a method of TRichView, but SysUtils.Format)

Note: TRichView allows empty text strings only in empty paragraphs (may be with list makers). See Valid documents 74.

Parameters:

FormatStr – format string. See "Format function" in the Delphi help file for syntax. **FormatStr** must not contain CR, LF and TAB characters. To add several lines of text use AddTextNL 171.

Args – arguments for format string.

StyleNo is an index in the TextStyles 404 collection of the linked 156 RVStyle component, or rvsDefStyle 663 constant. It defines font attributes for the text.

If **ParaNo**=-1, the methods add the item to the end of the last paragraph. If **ParaNo**>=0, this item starts a new paragraph with the **ParaNo**-th style. (**ParaNo** is an index in the ParaStyles 398 collection of the linked 156 RVStyle component). It defines attributes for the new paragraph.

Method type: viewer-style.

Unicode note: the parameter of this method is Unicode string for Delphi/C++Builder 2009 or newer, and ANSI string for older versions of Delphi/C++Builder.

Example:

```
MyRichView.AddFmt('X=%d and Y=%d', [X,Y], 0, 0);
```

If style templates 158 are used, the component may automatically change text style indices of text items, to provide a consistency of text and paragraph style templates.

See also methods:

- AddNL 167;
- AddTextNL 171;
- Format 182;
- FormatTail 183;
- SetAddParagraphMode 225.

See also properties of TRVStyle:

- TextStyles 404;
- ParaStyles 398;
- SpacesInTab 402.

See also:

- Other methods for appending items 43.

1.4.1.1.3.6 TRichView.AddHotPicture and AddHotPictureTag

These method add picture-hyperlink 93 (**gr**) with the specified **Name** and vertical alignment (**VAlign**) to the end of document.

```
procedure AddHotPictureTag(const Name: TRVAnsiString 623;
  gr: TGraphic; ParaNo: Integer;
```

```
  VAlign: TRVVAlign[648]; const Tag: TRVTag[645]);

procedure AddHotPicture(const Name: TRVAnsiString[623];
  gr: TGraphic; ParaNo: Integer; VAlign: TRVVAlign[648]);
```

(introduced in version 1.5)

AddHotPicture provides a subset of functionality of **AddHotPictureTag**:

- AddHotPicture(...) is equivalent to AddHotPictureTag(..., '')

The same as AddPictureEx and AddPictureExTag[169], but this picture will be a hypertext link.

Parameters:

Name – name of this *hot-picture* item, any string. **Name** must not contain CR and LF characters. RichView does not use Names itself, they are for your own use. This is an ANSI string, regardless of Unicode mode of text styles.

gr – picture to insert. By default, this picture will be owned by TRichView control, and you must not free it. However, you can specify nonzero *rvepShared* extra item property[627], and this graphic object will be shared (you need to free it yourself after TRichView is cleared).

VAlign – vertical align of this picture, relative to its line, see TRVVAlign[648] for possible values.

If **ParaNo**=-1, the methods add the item to the end of the last paragraph. If **ParaNo**>=0, this item starts a new paragraph with the **ParaNo**-th style. (**ParaNo** is an index in the ParaStyles[398] collection of the linked[156] RVStyle component). It defines attributes for the new paragraph.

Tag – *tag*[34] of this *hot-picture* item..

Alternatives:

- If you want to add a simple (not a hyperlink) TGraphic picture, add a picture[169] instead.
- If you want to add a hyperlink-picture from image list, add a [164]*hotspot*[164] instead.
- If you want to add a simple (not a hyperlink) picture from image list, add a [160]*bullet*[160] instead.

Methods type: viewer-style.

Unicode note: names of non-text items are ANSI strings.

See also methods:

- Format[182];
- FormatTail[183];
- SetAddParagraphMode[225].

See also methods of TRichViewEdit:

- InsertHotPicture[319].

See also properties of TRVStyle:

- ParaStyles[398].

See also:

- Item types[29];
- Other methods for appending items[43];
- "Tags"[34];
- Hypertext in RichView.

1.4.1.1.3.7 TRichView .AddHotspotExTag and Others

These methods add *hotspot*[97] (image from TImageList – hypertext link) to the end of document.

```
procedure AddHotspotExTag(const Name: TRVAnsiString[623];
  ImageIndex, HotImageIndex: Integer;
```

```
  ImageList: TCustomImageList; ParaNo: Integer;
  const Tag: TRVTag[645]);

procedure AddHotspotEx(const Name: TRVAnsiString[623];
  ImageIndex, HotImageIndex: Integer;
  ImageList: TCustomImageList; ParaNo: Integer);
procedure AddHotspot(ImageIndex: Integer;
  ImageList: TCustomImageList; fromnewline: Boolean); deprecated;
```

AddHotspotEx provides a subset of functionality of **AddHotspotExTag**:

- AddHotspotEx(...) is equivalent to AddHotspotExTag(..., '').

AddHotspot is deprecated and maintained for backward compatibility (with TRichView versions prior to 1.0):

- AddHotspot(ImageIndex, ImageList, True) is equivalent to AddHotspotEx('', ImageIndex, ImageIndex, ImageList, 0).
- AddHotspot(ImageIndex, ImageList, False) is equivalent to AddHotspotEx('', ImageIndex, ImageIndex, ImageList, -1).

Alternatives:

- If you want to add a simple (not a hyperlink) picture from image list, add a [160] *bullet* [160] instead.
- If you want to add a TGraphic picture-hyperlink, add a [163] *hot picture* [163] instead.
- If you want to add a TGraphic picture, add a picture [169] instead.

Parameters:

Name – name of *hotspot*, any string. **Name** must not contain CR and LF characters. RichView does not use Names itself, they are for your own use. This is an ANSI string, regardless of Unicode mode of text styles.

ImageIndex – index of image in **ImageList**.

HotImageIndex – index of "hot" image in **ImageList**. It is displayed when the mouse pointer is above this *hotspot*.

ImageList – image list for this *hotspot*. RichView does not own, does not copy and does not destroy imagelists, it just holds pointers to them. So this image list is not destroyed when the document is cleared.

If **ParaNo**=-1, the methods add the item to the end of the last paragraph. If **ParaNo**>=0, this item starts a new paragraph with the **ParaNo**-th style. (**ParaNo** is an index in the ParaStyles [398] collection of the linked [156] RVStyle component). It defines attributes for the new paragraph.

Tag – *tag* [34] of this *hotspot*.

Methods type: viewer-style.

Unicode note: names of non-text items are ANSI strings.

See also methods:

- Format [182];
- FormatTail [183];
- SetAddParagraphMode [225].

See also methods of TRichViewEdit:

- InsertHotspot [319].

See also properties of TRVStyle:

- ParaStyles [398].

See also:

- Item types [29];
- Other methods for appending items [43];
- "Tags" [34];
- Hypertext in RichView [35].

1.4.1.1.3.8 TRichView .AddItem

Adds one item to the end of document.

```
procedure AddItem(const Text: String; Item: TCustomRVItemInfo[526]);
```

(introduced in version 1.4)

This is the most general method for appending items to the document. Using this method you can add items of custom types.

For items of standard types it's recommended to use other methods for appending items[43].

Usually this method is used for appending tables[98], labels[101], sequences[102], footnotes[104], endnotes[103], references to notes[108].

The detailed description of item types is beyond the scope of this help file. Briefly, standard item types are declared and implemented in RVItem unit. All item types are classes derived from one base class – TCustomRVItemInfo[526].

Parameters:

Text – item name, some text associated with this item. It must not contain CR and LF characters.

Item – the item itself. **Item**.ParaNo defines the paragraph style of item (index in Style.ParaStyles[398])

Method type: viewer-style.

Unicode note: for Delphi/Unicode 2009 or newer, Text is Unicode string; for the older versions of Delphi/C++Builder, Text is ANSI string. It is converted to ANSI/Unicode, if necessary. Names of non-text items are ANSI strings.

See also methods:

- Format[182];
- FormatTail[183];
- SetAddParagraphMode[225];
- GetItem[190].

See also methods of TRichViewEdit:

- InsertItem[320].

See also:

- Item types[29];
- Other methods for appending items[43].

1.4.1.1.3.9 TRichView .AddNamedCheckpointExTag and Others

These methods add *checkpoint*[30] to the end of document.

```
function AddNamedCheckpointExTag(const CpName: String;
  RaiseEvent: Boolean; const Tag: TRVTag[645]): Integer;

function AddCheckpoint: Integer;
function AddCheckpointTag(const Tag: TRVTag[645]): Integer;
function AddNamedCheckpoint(const CpName: String): Integer;
function AddNamedCheckpointEx(const CpName: String;
  RaiseEvent: Boolean): Integer;
```

AddCheckpoint, AddCheckpointTag, AddNamedCheckpoint, AddNamedCheckpointEx provide subsets of functionality of **AddNamedCheckpointExTag**:

- AddCheckpoint is equivalent to AddNamedCheckpointExTag('', *False*, '')..
- AddCheckpointTag(Tag) is equivalent to AddNamedCheckpointExTag('', *False*, Tag).
- AddNamedCheckpoint(CpName) is equivalent to AddNamedCheckpointExTag(CpName, *False*, '').

- AddNamedCheckpointEx(CpName, RaiseEvent) is equivalent to AddNamedCheckpointExTag(CpName, RaiseEvent, '').

Checkpoints are not items, but the methods for appending *checkpoints* are similar to methods for appending items. This *checkpoint* will be associated with the next appended item.

Parameters:

CpName – name of *checkpoint*. This is an ANSI string, regardless of Unicode mode of text styles. It must not contain #0, #1 and #2 characters.

RaiseEvent – "raise event" flag, see OnCheckpointVisible 238

Tag – *tag* of *checkpoint*.

Return value:

Index of the added checkpoint

Methods type: viewer-style.

Unicode note: names of *checkpoints* are Unicode strings for Delphi/C++Builder 2009 or newer, and ANSI strings for older versions of Delphi/C++Builder.

See also methods:

- SetCheckpointInfo 227;
- RemoveCheckpoint 214.

See also methods of TRichViewEdit:

- SetCheckpointInfoEd 340;
- SetCurrentCheckpointInfo 343.

See also:

- Methods for appending items 43;
- *Checkpoints* 30;
- *Tags* 34.

1.4.1.1.3.10 TRichView.AddNLTag -A- -W-

These methods add one text item 88 with the **StyleNo**-th text style and the **ParaNo**-th paragraph style, with the specified **Tag** to the end of document.

Methods with the full set of parameters:

```
procedure AddNLTag(s: String; StyleNo, ParaNo: Integer; const Tag: TRVTag 645);
procedure AddNLATag(const s: TRVAnsiString 623; StyleNo, ParaNo: Integer; const Tag: TRVTag
procedure AddNLWTag(const s: TRVUnicodeString 647; StyleNo, ParaNo: Integer; const Tag: TRV
```

Methods with a shortened set of parameters:

```
procedure AddNL(const s: String; StyleNo, ParaNo: Integer);
procedure Add(const s: String; StyleNo:Integer);
procedure AddTag(const s: String; StyleNo:Integer; const Tag: TRVTag 645);
```

(introduced in version 1.3 and 1.4)

Parameters:

s is a text string to add. It must not contain CR, LF, TAB, FF characters (#13, #10, #9, #12). To add several lines of text use AddTextNL 171.

StyleNo is an index in the TextStyles 404 collection of the linked 156 RVStyle component, or *rvsDefStyle* 663 constant. It defines font attributes for the text.

If **ParaNo**=-1, the methods add the item to the end of the last paragraph. If **ParaNo**>=0, this item starts a new paragraph with the **ParaNo**-th style. (**ParaNo** is an index in the ParaStyles 398 collection of the linked 156

RVStyle component). It defines attributes for the new paragraph.

Tag – *tag* [34] of this text item.

AddNL, **Add**, **AddTag** provide subsets of functionality of **AddNLTag**:

- AddNL(...) is equivalent to AddNLTag(..., '').
- Add(...) is equivalent to AddNLTag(..., -1, '').
- AddTag(s, StyleNo, Tag) is equivalent to AddNLTag(s, StyleNo, -1, Tag).

In other words:

- **AddNL** adds text string as a beginning of new paragraph or to the end of the last paragraph.
- **AddNLTag** adds text string with tag as a beginning of new paragraph or to the end of the last paragraph.
- **Add** adds text string to the end of the last paragraph.
- **AddTag** adds text string with tag to the end of the last paragraph.

Note: TRichView allows empty text strings only in empty paragraphs (may be with list makers). See Valid documents [74].

Unicode notes:

AddNLATag:

- If the **StyleNo**-th style is ANSI style, **s** is added as it is.
- If the **StyleNo**-th style is Unicode style, **s** is converted to Unicode (conversion is performed basing on the Charset [434] of this style).

AddNLWTag:

- If the **StyleNo**-th style is Unicode style, **s** is added as it is.
- If the **StyleNo**-th style is ANSI style, s is converted to ANSI (conversion is performed basing on the Charset [434] of this style).

AddNLTag works like **AddNLWTag** in Delphi/C++Builder 2009, and like **AddNLATag** in the older versions of Delphi/C++Builder.

Methods type: viewer-style

If style templates [158] are used, the component may automatically change text style indices of text items, to provide a consistency of text and paragraph style templates.

See also methods:

- AddTab [170];
- Format [182];
- FormatTail [183];
- SetAddParagraphMode [225].

See also methods of TRichViewEdit:

- InsertStringTag, InsertStringATag, InsertStringWTag [326].

See also properties of TRVStyle:

- TextStyles [404];
- SpacesInTab [402].

See also:

- Other methods for appending items [43];
- Unicode in RichView.

1.4.1.1.3.11 TRichView.AddPictureExTag and Others

These methods add picture (90) (**gr**) with the specified **Name** and vertical alignment (**VAlign**) to the end of document

```
procedure AddPictureExTag(const Name: TRVAnsiString(623); gr: TGraphic;
  ParaNo: Integer; VAlign: TRVVAlign(648); const Tag: TRVTag(645));
procedure AddPictureEx(const Name: TRVAnsiString(623); gr: TGraphic;
  ParaNo: Integer; VAlign: TRVVAlign(648));
procedure AddPicture(gr: TGraphic); deprecated;
```

AddPictureEx provides a subset of functionality of **AddPictureExTag**:

- AddPictureEx(...) is equivalent to AddPictureExTag(..., '').

The same as AddHotPicture and AddHotPictureTag (163), but this picture will not be a hypertext link.

AddPicture is deprecated and maintained for backward compatibility (with TRichView versions prior to 1.0):

- AddPicture(gr) is equivalent to AddPictureEx('', gr, 1, rvvaBaseline).

Parameters:

Name – name of this picture item, any string. **Name** must not contain CR and LF characters. RichView does not use Names itself, they are for your own use. This is an ANSI string, regardless of Unicode mode of text styles.

gr – picture to insert. By default, this picture will be owned by TRichView control, and you must not free it. However, you can specify nonzero *rvepShared* extra item property (627), and this graphic object will be shared (you need to free it yourself after TRichView is cleared).

VAlign – vertical align of this picture, relative to its line, see TRVVAlign (648) for possible values.

If **ParaNo**=-1, the methods add the item to the end of the last paragraph. If **ParaNo**>=0, this item starts a new paragraph with the **ParaNo**-th style. (**ParaNo** is an index in the ParaStyles (398) collection of the linked (156) RVStyle component). It defines attributes for the new paragraph.

Tag – *tag* (34) of this picture item.

Alternatives:

- If you want to add a TGraphic picture-hyperlink, add a (163) *hot picture* (163) instead.
- If you want to add a hyperlink-picture from image list, add a (164) *hotspot* (164) instead.
- If you want to add a simple (not a hyperlink) picture from image list, add a (160) *bullet* (160) instead.

Methods type: viewer-style.

Unicode note: names of non-text items are ANSI strings.

Example 1:

```
procedure AddBitmapFromFile(rv: TCustomRichView;
  const FileName: String; ParaNo: Integer);
var bmp: TBitmap;
begin
  bmp := TBitmap.Create;
  bmp.LoadFromFile(FileName);
  rv.AddPictureEx('', bmp, ParaNo, rvvaBaseline);
end;
...
// creating document with 2 pictures
// in the same paragraph
rv.Clear(175);
AddBitmapFromFile(rv, 'd:\pic1.bmp', 0);
AddBitmapFromFile(rv, 'd:\pic2.bmp', -1);
```

```
rv.Format[182];
```

Example 2: adding shared images

```
procedure AddBitmapFromFile(rv: TCustomRichView;
  const FileName: String; ParaNo: Integer);

bmp := TBitmap.Create;
bmp.LoadFromFile('c:\pic.bmp');
rv.AddPictureEx('', bmp, 0, rvvaBaseline);
rv.SetItemExtraIntProperty[230](rv.ItemCount[144]-1, rvepShared, 1);
rv.AddPictureEx('', bmp, -1, rvvaBaseline);
rv.SetItemExtraIntProperty[230](rv.ItemCount[144]-1, rvepShared, 1);
rv.Format[182];
// you must free bmp yourself when rv is cleared.
```

See also methods:

- Format[182];
- FormatTail[183];
- SetAddParagraphMode[225].

See also methods of TRichViewEdit:

- InsertPicture[322].

See also properties of TRVStyle:

- ParaStyles[398].

See also:

- Item types[29];
- Other methods for appending items[43];
- "Tags"[34].

1.4.1.1.3.12 TRichView .AddTab

Adds tabulator[89] to the end of document.

```
procedure AddTab(TextStyleNo, ParaNo: Integer);
```

(introduced in v1.9)

Parameters:

TextStyleNo – text style of tabulator. It is an index in the TextStyles[404] collection of the linked[156] RVStyle component, or *rvsDefStyle*[663] constant. It defines font attributes.

If **ParaNo**=-1, the methods add the item to the end of the last paragraph. If **ParaNo**>=0, this item starts a new paragraph with the **ParaNo**-th style. (**ParaNo** is an index in the ParaStyles[398] collection of the linked[156] RVStyle component). It defines attributes for the new paragraph.

This method always adds a tabulator, regardless of RVStyle.SpacesInTab[402] mode. If RVStyle.SpacesInTab[402] =0, this method does the same work as AddTextNL[171](#9, TextStyleNo, ParaNo, ParaNo).

Methods type: viewer-style.

See also methods:

- Format[182];
- FormatTail[183];
- SetAddParagraphMode[225].

See also methods of TRichViewEdit:

- InsertTab[326].

See also properties of TRVStyle:

- ParaStyles[398].

See also:

- Item types 29;
- Other methods for appending items 43.

1.4.1.1.3.13 TRichView.AddText, AddTextFromNewLine (deprecated)

Obsolete, use AddTextNL 171

```
procedure AddText(const s: String; StyleNo:Integer); deprecated;
procedure AddTextFromNewLine(const s: String; StyleNo:Integer); deprecated;
```

These methods are deprecated and maintained for backward compatibility (with TRichView versions prior to 1.0).

AddText(...) is equivalent to AddTextNL 171(..., -1, 0).

AddTextFromNewLine(...) is equivalent to AddTextNL 171(..., 0, 0).

Methods type: viewer-style.

1.4.1.1.3.14 TRichView.AddTextNL -A -W

These methods add one or more text 88 lines to the end of document.

```
procedure AddTextNL(s: String; StyleNo,
  FirstParaNo, OtherParaNo: Integer);
procedure AddTextNLA(const s: TRVAnsiString 623; StyleNo,
  FirstParaNo, OtherParaNo: Integer);
procedure AddTextNLW(const s: TRVUnicodeString 647;
  StyleNo, FirstParaNo, OtherParaNo: Integer;
  DefAsSingleParagraph: Boolean);
```

Parameters:

s is a text string to add. It may contain special characters:CR, LF, TAB, FF (#13, #10, #9, #12). #9 characters may be inserted as tabulators 89, depending on value of SpacesInTab 402 property of the linked 156 RVStyle component. #12 characters add page breaks 149. All possible line breaks modes (CR, LF, CR+LF, LF+CR) are supported.

StyleNo is an index in the TextStyles 404 collection of the linked 156 RVStyle component, or *rvsDefStyle* 663 constant. It defines font attributes for the text.

ParaNo defines paragraph attributes for the fist line of the added text. If **ParaNo**=-1, the methods add the first line to the end of the last paragraph. If **ParaNo**>=0, this line starts a new paragraph with the **ParaNo**-th style. (**ParaNo** is an index in the ParaStyles 398 collection of the linked 156 RVStyle component).

OtherParaNo defines paragraph attributes for the subsequent lines of text. Must be >=0.

DefAsSingleParagraph: If *False*, **AddTextNLW** uses the current add-paragraph mode (see SetAddParagraphMode 225). If *True*, it treats #13 and #10 as line breaks, not paragraph breaks. WideChar($2029) ("paragraph separator") always initiates a new paragraph. WideChar($2028) ("line separator") always initiates a new line.

AddTextNLW supports the Unicode byte order marks characters (if it's present, it must be the first character in **s**).

Methods type: viewer-style.

Unicode notes:

AddTextNLA:

- If the **StyleNo**-th style is ANSI style, **s** is added as it is.
- If the **StyleNo**-th style is Unicode style, **s** is converted to Unicode (conversion is performed basing on the Charset 434 of this style).

AddTextNLW:

- If the **StyleNo**-th style is Unicode style, **s** is added as it is..
- If the **StyleNo**-th style is ANSI style, s is converted to ANSI (conversion is performed basing on the Charset 434 of this style).

AddTextNL: works like **AddTextNLW** in Delphi/C++Builder 2009 or newer, and like **AddTextNLA** in the older versions of Delphi/C++Builder.

If style templates 158 are used, the component may automatically change text style indices of text items, to provide a consistency of text and paragraph style templates.

See also methods:

- AddNL and others 167;
- AddTab 170;
- LoadText, LoadTextW 212;
- LoadTextFromStream, LoadTextFromStreamW 213;
- Format 182;
- FormatTail 183;
- SetAddParagraphMode 225.

See also methods of TRichViewEdit:

- InsertText, -A, -W 327.

See also properties of TRVStyle:

- TextStyles 404;
- SpacesInTab 402.

See also:

- Other methods for appending items 43;
- Unicode in RichView 64;
- Example how to load UTF-8 files 282.

1.4.1.1.3.15 TRichView .AppendFrom

Adds content of another RichView component (**Source**) to the end of document.

```
procedure AppendFrom(Source: TCustomRichView 128);
```

This method does not copy items of some types (including inserted controls 94 and tables 98) (it just ignores them).

This method is the fastest way to copy items from one TRichView to another, but it has the limitations listed above.

The recommended way of copying is:

```
var Stream: TMemoryStream;
begin
  Stream := TMemoryStream.Create;
  try
    Source.SaveRVFToStream 220(Stream, False);
    Stream.Position := 0;
    Dest.LoadRVFFromStream 211(Stream);
    Dest.Format 182;
  finally
    Stream.Free;
  end;
end;
```

Method type: viewer-style.

See also methods:

- Format 182;
- FormatTail 183.

See also:

- Other methods for appending items 43.

1.4.1.1.3.16 TRichView.AppendRVFFromStream

Adds data from Stream in RichView Format (RVF) 59 to the end of document.

```
function AppendRVFFromStream(Stream: TStream;
  ParaNo: Integer): Boolean;
```

This method appends content from RVF stream. It ignores the paragraph style of first paragraph in RVF and uses the **ParaNo**-th paragraph style instead of it.

Parameters:

Stream – stream containing document in RichView Format (RVF).

If **ParaNo**=-1, the method adds the first item to the end of the last paragraph. If **ParaNo**>=0, this item starts a new paragraph with the **ParaNo**-th style. (**ParaNo** is an index in the ParaStyles 398 collection of the linked 156 RVStyle component). This parameter affects only the first paragraph of inserted data. Styles of other paragraphs are defined in RVF.

Method type: viewer-style.

Setting for RVF loading can be changed in the TRichView component editor 134.

If style templates are used 158, and RVFTextStylesReadMode 154=RVFParaStylesReadMode 154 =*rvf_sInsertMerge*, and StyleTemplateInsertMode 156<>*rvstimIgnoreSourceStyleTemplates*, the method merges style templates from RVF into Style 156.StyleTemplates 402, and reads text and paragraph styles according to StyleTemplateInsertMode 156. The method calls OnStyleTemplatesChange 266 event.

Return value:

"Was reading successful?"

See also properties:

- RVFOptions 152;
- RVFWarnings 155;
- RVFTextStylesReadMode 154 (this method type: RVF insertion);
- RVFParaStylesReadMode 154 (this method type: RVF insertion);
- UseStyleTemplates 158.

See also events:

- OnRVFImageListNeeded 252;
- OnRVFControlNeeded 251;
- OnRVFPictureNeeded 252;
- OnControlAction 239 (ControlAction=*rvcaAfterRVFLoad*);
- OnStyleTemplatesChange 266.

See also methods:

- InsertRVFFromStream 206;
- LoadRVFFromStream 211;
- LoadRVF 211;
- Format 182;
- FormatTail 183.

See also:

- TRichView component editor 134;
- Saving and loading 58;
- RVF overview 59;
- Paragraphs 37.

1.4.1.1.3.17 TRichView .AssignSoftPageBreaks

Displays soft page breaks (i.e. page breaks that were made automatically)

```
procedure AssignSoftPageBreaks(RVPrint[479]: TComponent);
```

(introduced in version 1.7)

Page breaks become visible after calling RichView.AssignSoftPageBreaks(RVPrint). **RVPrint** must be formatted (i.e. FormatPages[490] must be called before this method). Page breaks are visible only if *rvoShowPageBreaks* is included in RichView.Options[146].

In editor[282], soft page breaks will be automatically cleared after any editing operation (any action invoking OnChange[357]). Page breaks will also be cleared when you call Clear[175]. In all other cases, you must call ClearSoftPageBreaks[176] after each change in the document performed after calling AssignSoftPageBreaks.

Soft page breaks are drawn using RVStyle.SoftPageBreakColor[401] (or in OnDrawPageBreak[413] event).

See also:

- ClearSoftPageBreaks[176].

See also properties:

- PageBreaksBeforeItems[149].

1.4.1.1.3.18 TRichView .BeginOleDrag

Starts dragging the selection.

```
procedure BeginOleDrag;
```

(introduced in v1.9)

This method is useful if you want to implement drag&drop[55] of inserted controls[94].(all other content can be dragged automatically)

Example

This example implements drag&drop of inserted TButton controls.

1) Add the following variables in form:

```
ClickedControl: TObject;
ClickPoint: TPoint;
```

2) Add the following methods:

```
procedure TForm1.OnControlMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  if Button = mbLeft then begin
    RichViewEdit1.SelectControl[223](TControl(Sender));
    ClickedControl := Sender;
    ClickPoint := Point(X, Y);
  end;
end;

procedure TForm1.OnControlMouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin
  if (ssLeft in Shift) and (ClickedControl=Sender) and
    (Sqr(ClickPoint.X-X)+Sqr(ClickPoint.Y-Y)>10) then
    RichViewEdit1.BeginOleDrag;
end;
```

3) Assign events to created controls:

```
  btn := TButton.Create(Self);
  btn.Caption := 'Button';
```

```
  btn.OnMouseDown := OnControlMouseDown;
  btn.OnMouseMove := OnControlMouseMove;
  RichViewEdit1.InsertControl(318)('',btn,rvvaBaseline);
```

4) Events are not saved in RVF, so reassign them on loading. Use OnControlAction(239) event:

```
procedure TForm1.RichViewEdit1ControlAction(Sender: TCustomRichView;
  ControlAction: TRVControlAction; ItemNo: Integer; var ctrl: TControl);
begin
  if ControlAction=rvcaAfterRVFLoad then begin
    if ctrl is TButton then begin
      TButton(ctrl).OnMouseDown := OnControlMouseDown;
      TButton(ctrl).OnMouseMove := OnControlMouseMove;
     end;
  end;
end;
```

The complete example can be downloaded here: ***http://www.trichview.com/forums/viewtopic.php?t=157***

1.4.1.1.3.19 TRichView .Clear

Clears the document.

```
procedure Clear; virtual;
```

This method:

- deletes all items(29);
- turns off displaying soft page breaks(176);
- clears DocProperties(141);
- stops live spelling check(175);
- stops animations(56), if AnimationMode(135) = *rvaniOnFormat*.

This method does not change collections of text, paragraph and list styles. So, if your application adds styles while editing, call DeleteUnusedStyles(181) after Clear.

Method type: generally, it's a viewer-style method, but it is special, because it clears all editing and displaying history. It is a starting method for document generation.

This method is called by the methods: LoadRVF(211), LoadRVFFromStream(211).

See also methods:

- Format(182).

See also method of TRichViewEdit:

- Clear(304).

See also:

- Methods for building documents(43).

1.4.1.1.3.20 TRichView .ClearLiveSpellingResults

Clears all live spelling underlines and stops the live spelling thread.

```
procedure ClearLiveSpellingResults;
```

(introduced in v1.9)

Clear(175) method calls this procedure automatically.

See also: Live spelling check in RichView(69).

1.4.1.1.3.21 TRichView .ClearSoftPageBreaks

Turns off displaying soft page breaks

```
procedure ClearSoftPageBreaks;
```

See also:

- AssignSoftPageBreaks [174].

1.4.1.1.3.22 TRichView .ClientToDocument, DocumentToClient

The methods convert client coordinates to document coordinates and vice versa.

```
function ClientToDocument(const APoint: TPoint): TPoint;
```

(introduced in v1.9)

```
function DocumentToClient(const APoint: TPoint): TPoint;
```

(introduced in v14)

ClientToDocument converts **APoint** from the control's coordinate system to the document coordinate system. DocumentToClient performs an opposite conversion.

In client area coordinates, (0, 0) corresponds to the top left corner of the control's client area. For example, mouse events use this coordinate system.

In document coordinates, (0, 0) corresponds to the top left corner of the scrollable area. For example, GetItemAt [190] uses this coordinate system.

The methods take a cell rotation [573] into account.

Note:

This method must be used instead of adding HScrollPos [510] to X and VScrollPos [512]*VSmallStep [159] to Y. The old method was used in some old example code, it is correct only if the document is top-aligned [159].

See also:

Example in OnRVMouseDown [253].

1.4.1.1.3.23 TRichView .ConvertDocTo...

The methods convert all lengths in the current document to different units of measurement.

```
procedure ConvertDocToPixels;
procedure ConvertDocToTwips;
procedure ConvertDocToDifferentUnits(NewUnits: TRVStyleUnits [644]);
```

(introduced in version 13)

The methods convert all lengths in the current document from Style [156].Units [404] to different units of measurement:

- screen pixels [74],
- twips [74],
- NewUnits (twips or pixels),

respectively.

The methods convert only properties of items [85], they do not convert properties of the Style [156] itself.

To convert the document completely, you need to:

- call one of these methods (for all documents linked to the same TRVStyle [385] components);
- call one of Style [156].ConvertTo* [405] method to convert properties of Style [156] to the same units of measurement.

1.4.1.1.3.24 TRichView.Copy

Copies the selected fragment to the Clipboard in all available formats (as text, RVF, RTF and picture)

```
procedure Copy;
```

This procedure does nothing if there is nothing selected.

This procedure clears the Clipboard before copying.

This method must be called only when the document is formatted.

See also:

- Working with selection 47;
- Working with Clipboard 48.

1.4.1.1.3.25 TRichView.CopyDef

Copies the selected fragment to the Clipboard in several formats according to Options 146.

```
function CopyDef: Boolean;
```

This is the main method for copying to the Clipboard. It is executed automatically when user presses Ctrl+C or Ctrl+Insert.

Calling CopyDef does the same thing as sending the control a WM_COPY message.

This procedure does nothing if there is nothing selected.

This procedure clears the Clipboard before copying.

- if (*rvoAutoCopyRVF* in Options), it copies selection as RVF 59.
- if (*rvoAutoCopyText* in Options), it copies selection as ANSI text. ANSI text is not copied under Windows NT based systems (Windows 2000, XP, Vista), if *rvoAutoCopyUnicodeText* is included in Options. In Windows NT, the Clipboard can convert ANSI and Unicode texts to each other.
- if (*rvoAutoCopyImage* in Options), and there is only one picture selected, it copies selection as an image.
- if (*rvoAutoCopyUnicodeText* in Options), it copies text as Unicode 64 text.
- if (*rvoAutoCopyRTF* in Options), it copies selection as RTF.

This method must be called only when the document is formatted.

Return value:

"Is the selection not empty?" and "Is at least one of *rvoAutoCopy*** options set?"

See also:

- Working with selection 47;
- Working with Clipboard 48.

1.4.1.1.3.26 TRichView.CopyImage

Copies the selected image to the Clipboard.

```
procedure CopyImage;
```

This procedure does nothing if there is nothing selected.

This procedure clears the Clipboard before copying.

If selection contains something besides one image, this method just clears the Clipboard.

This method must be called only when the document is formatted.

See also:

- Working with selection 47;
- Working with Clipboard 48.

1.4.1.1.3.27 TRichView .CopyRTF

Copies the selected part of the document to the Clipboard as RTF (Rich Text Format).

```
procedure CopyRTF;
```

This procedure does nothing if there is nothing selected.

This procedure clears Clipboard before copying.

Settings for RTF saving can be changed in the TRichView component editor 134.

Options for saving are in RTFOptions 150 property.

By default, items tags 34 are saved as hyperlinks targets. You can customize saving using OnWriteHyperlink 267 event.

This method must be called only when the document is formatted.

See also:

- SaveRTF 219, SaveRTFToStream 220 methods;
- RTFOptions 150 property;
- UseStyleTemplates 158 property;
- Working with selection 47;
- Working with Clipboard 48.

1.4.1.1.3.28 TRichView .CopyRVF

Copies the selection to the Clipboard as RVF (RichView Format)

```
procedure CopyRVF;
```

This procedure does nothing if there is nothing selected.

This procedure clears the Clipboard before copying.

Settings for RVF saving can be changed in the TRichView component editor 134.

This method must be called only when the document is formatted.

See also:

- Working with selection 47;
- Working with Clipboard 48;
- RVF 59.

1.4.1.1.3.29 TRichView .CopyText -W

These methods copy the selected fragment to the Clipboard as text.

```
procedure CopyText;
procedure CopyTextA;
procedure CopyTextW;
```

These methods do nothing if there is nothing selected.

These methods clear the Clipboard before copying.

CopyTextA copies text in ANSI encoding. Text of Unicode styles is converted basing on TRVStyle.DefCodePage 388.

CopyTextW copies text in Unicode encoding. ANSI strings are converted basing on the corresponding Charsets 434. For text of DEFAULT_CHARSET, TRVStyle.DefCodePage 388 is used.

CopyText works like **CopyTextW** in Delphi/C++Builder 2009 or newer, and like **CopyTextA** in the older versions of Delphi/C++Builder.

These methods must be called only when the document is formatted.

See also:

- Working with selection 47;
- Working with Clipboard 48;
- Unicode 64.

1.4.1.1.3.30 TRichView.Create

Usual component constructor. Creates a new TRichView instance.

```
constructor Create(AOwner: TComponent);override;
```

Use Create to instantiate RichView at runtime. RichViews placed on forms at design time are created automatically. Specify the owner of the new RichView using the **AOwner** parameter.

Important note: initial values of several properties of components created in code may be different from initial values of components placed on form at design time. In the latter case, initial values of these properties can be overridden by the designtime component editor 134 (and overridden by default!)

1.4.1.1.3.31 TRichView.DeleteItems

Removes the specified number of items (**Count**) from the document, starting from the **FirstItemNo**-th item.

```
procedure DeleteItems(FirstItemNo, Count: Integer);
```

Items are indexed from 0 to ItemCount 144-1. Items of subdocuments (table cells 567) are not included in the items range of the main document; for items in cells, use Cell.GetRVData 575.DeleteItems.

Method type: viewer-style.

Be careful: this method can produce invalid document 74.

Example 1: deleting all empty lines

```
procedure DeleteBlankLines(RVData: TCustomRVData 608);
var i,r,c: Integer;
    table: TRVTableItemInfo 528;
begin
  for i := RVData.ItemCount 144-1 downto 0 do
    if RVData.IsParaStart 208(i) and (RVData.GetItemStyle 194(i)>=0) and
      (RVData.GetItemTextR(i)='') and (RVData.ItemCount 144>1) then
      RVData.DeleteItems(i,1)
    else if RVData.GetItemStyle 194(i)=rvsTable then begin
      table := TRVTableItemInfo 528(RVData.GetItem 190(i));
      for r := 0 to table.RowCount 543-1 do
        for c := 0 to table.ColCount 538-1 do
          if table.Cells 537[r,c]<>nil then
            DeleteBlankLines(table.Cells 537[r,c].GetRVData 575);
    end;
end;
```

DeleteBlankLines cannot be undone/redone. If called for editor, the editor's undo buffer becomes incorrect, so call ClearUndo 305.

Call:

```
DeleteBlankLines(RichViewEdit1.RVData 151);
RichViewEdit1.ClearUndo 305;
RichViewEdit1.Format 182;
```

Example 2: deleting empty lines at the end of document

```
procedure DeleteTrailingBlankLines(RVData: TCustomRVData);
var i: Integer;
begin
  for i := RVData.ItemCount-1 downto 1 do
```

```
    if RVData.IsParaStart(i) and (RVData.GetItemStyle(i)>=0) and
     (RVData.GetItemTextR(i)='') then
      RVData.DeleteItems(i,1)
    else
      break;
end;
```

See also:

- DeleteSection[180];
- DeleteParas[180].

1.4.1.1.3.32 TRichView.DeleteMarkedStyles

Deletes unused text[404], paragraph[398] and list[396] styles.

```
procedure MarkStylesInUse(Data: TRVDeleteUnusedStylesData[612]);
```

(introduced in version 10)

This method is useful if collections of text, paragraph and list styles are not saved in RVF documents[59] (but stored in file or in the Registry). In this case, several documents share the same collections of styles. Used styles are marked with MarkStylesInUse[213] method, see TRVDeleteUnusedStylesData[612] for the examples.

Usually it's much more simple (and recommended) to use one RVStyle per one RichView and store collections of styles inside RVF documents. In this case, use DeleteUnusedStyles[181] instead.

See also methods:

- MarkStylesInUse[213];
- DeleteUnusedStyles[181].

1.4.1.1.3.33 TRichView.DeleteParas

Deletes paragraph(s) containing items from **FirstItemNo** to **LastItemNo**, updates the document

```
procedure DeleteParas(FirstItemNo, LastItemNo: Integer);
```

(introduced in version 1.7)

This method is designed specially for chat/log windows. Together with FormatTail[183], it allows updating chat windows faster, without complete reformatting.

Items are indexed from 0 to ItemCount[144]-1..

Method type: unique method. Unlike all other methods of TRichView, it quickly reformats RichView, so you do not need to call Format[182] after it. Like editing-style methods, this method requires document to be formatted before calling, and leaves it formatted after. But it is still a viewer-style method, because it does not update undo buffers, does not check ReadOnly property, etc.

See also:

- DeleteItems[179];
- DeleteSection[180].

1.4.1.1.3.34 TRichView.DeleteSection

Removes the "section" named **CpName** from documents

```
procedure DeleteSection(CpName: String);
```

"Section" is a part of document. The first item of "section" is the item having *checkpoint*[30] with non empty name. This name is also a name of "section". If there is no another *checkpoint* with non empty name below, the last item in the document is also the last item of "section". Otherwise, item with that "checkpoint" starts a new

section, and does not belong to the current section.

Method type: viewer-style.

Be careful: this method can produce invalid document 74.

See also:

- DeleteItems 179;
- DeleteParas 180.

1.4.1.1.3.35 TRichView.DeleteUnusedStyles

Deletes text and/or paragraph and/or list styles which do not exist in document

```
procedure DeleteUnusedStyles(
  TextStyles, ParaStyles, ListStyles: Boolean);
```

(introduced in version 1.5, updated in version 1.6, 1.7)

This procedure deletes unused items from the collections of text (if **TextStyles** parameter is True), paragraph (if **ParaStyles** parameter is True), and list (if **ListStyles** parameter is True) styles (see TextStyles 404, ParaStyles 398, ListStyles 396) of the linked 156 TRVStyle component).

Then it updates references to these styles in the component and other styles.

Warning: be careful when using this method. Use it only if RVStyle object is used exclusively by this RichView (this method updates references to styles only in one RichView)

This method deletes only non-standard styles (see Standard 431 property of styles)

This method does not require reformatting.

See also methods:

- MarkStylesInUse 213;
- DeleteMarkedStyles 180.

1.4.1.1.3.36 TRichView.Deselect

Removes the document selection (does not change document, just nothing becomes selected for copying to the Clipboard)

```
procedure Deselect;
```

This method does not repaint document (call Invalidate)

This method must be called only when the document is formatted.

The method generates OnSelect 265 event.

See also methods:

- SetSelectionBounds 235;
- SelectAll 223.

See also:

- Selecting in RichView document 47.

1.4.1.1.3.37 TRichView.Destroy

Destroys an instance of TRichView

```
destructor Destroy;override;
```

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the control is not nil, and only then calls Destroy.

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

1.4.1.1.3.38 TRichView.FindCheckpointByName

Returns the first *checkpoint* having the specified **Name**, or **nil** if there is no such *checkpoint*

```
function FindCheckpointByName(const Name: String): TCheckpointData;
```

Unicode note: names of *checkpoints* are Unicode strings for Delphi/C++Builder 2009 or newer, and ANSI strings for older versions of Delphi/C++Builder.

See also methods:

- FindCheckpointByTag;

See also:

- "Checkpoints".

1.4.1.1.3.39 TRichView.FindCheckpointByTag

Returns the first checkpoint having the specified tag, or nil if there is no such checkpoint

```
function FindCheckpointByTag(const Tag: TRVTag): TCheckpointData;
```

Example:

```
var  CheckpointData: TCheckpointData;
...
  CheckpointData := MyRichView.FindCheckpointByTag('Tag_String_Example');
```

See also methods:

- FindCheckpointByName.

See also:

- "Tags";
- "Checkpoints".

1.4.1.1.3.40 TRichView.FindControlItemNo

Returns an index of the RichView item which owns the control **actrl**, or -1 if **actrl** was not added in RichView.

```
function FindControlItemNo(actrl: TControl): Integer;
```

Note: This method contains some calculations inside (iterations through items, from the 0-th to the returned one)

See also:

- RichView item types.

1.4.1.1.3.41 TRichView.Format

Formats the document. Only formatted documents can be displayed or edited.

```
procedure Format;
```

If *rvoFormatInvalidate* in Options (default), this method also repaints RichView.

You need to call this method after you modified document (using any of viewer-style methods), its text or paragraph styles.

When resized, RichView is reformatted automatically.

Most of methods of RichViewEdit 282 automatically quickly reformat the affected part of the document, but you still need this method in RichViewEdit when you use methods inherited from RichView (viewer-style methods).

See also methods:
- FormatTail 183;
- Reformat 214.

See also properties:
- Options 146.

See also:
- Building document 43.

1.4.1.1.3.42 TRichView.FormatTail

Formats a part of document appended after the last call of Format 182 or FormatTail.

```
procedure FormatTail;
```

This method works properly only if new items were added from new line (as new paragraphs) after the last call of Format 182 or FormatTail.

It's much more efficient to reformat only a new part of the document than making a full reformatting.

This method is designed specially for chat/log windows. Together with DeleteParas 180, it allows updating chat windows faster, without complete reformatting.

If *rvoFormatInvalidate* in Options 146, this method also repaints RichView.

See also methods:
- Format 182;
- DeleteParas 180.

See also properties:
- Options 146;
- VScrollVisible 512.

See also:
- Appending items 43.

1.4.1.1.3.43 TRichView.GetBreakInfo

Returns main properties of item of *break* 93 type 93

```
procedure GetBreakInfo(ItemNo: Integer; var AWidth: TRVStyleLength 644;
  var AStyle: TRVBreakStyle 624; var AColor: TColor;
  var ATag: TRVTag 645);
```

Input parameter:

ItemNo – index of the item. The item must be of *break* 93 type (*rvsBreak* 663), otherwise the method raises ERichViewError 608 exception. Items are indexed from 0 to ItemCount 144-1, GetItemStyle 194 returns type of item. Items of subdocuments (table cells 567) are not included in the items range of the main document; for items in cells, use Cell.GetRVData 575.GetBreakInfo.

Output parameters:

AWidth – line width (or rectangle height). This value is measured in Style 156.Units 404.

AStyle – visual style of this *break*, see TRVBreakStyle[624] for possible values.

AColor – line color. If it is equal to *clNone*, Style[156].TextStyles[404][0].Color[435] is used.

ATag – *tag* of the item. Use SetItemTag[231] or SetBreakInfo[226] to change *tag* of item.

Instead of this method, you can use GetItemExtraIntPropertyEx[193] and GetItemTag[195] methods.

Additional item properties are returned by the methods GetItemExtraIntProperty[193] and GetItemExtraStrProperty[193].

See also methods:

- SetBreakInfo[226];
- GetItemStyle[194];
- GetItemExtraIntProperty[193];
- GetItemExtraStrProperty[193].

See also properties:

- ItemCount[144].

See also:

- Obtaining RichView items[50];
- Item types[29];
- *Tags*[34].

1.4.1.1.3.44 TRichView .GetBulletInfo

Returns main properties of item of *bullet*[96] type[96]

```
procedure GetBulletInfo(ItemNo: Integer; var AName: TRVAnsiString[623];
  var AImageIndex: Integer; var AImageList: TCustomImageList;
  var ATag: TRVTag[645]);
```

Input parameter:

ItemNo – index of the item. The item must be of *bullet*[96] or *hotspot*[97] type (*rvsBullet*[663] or *rvsHotspot*[663]), otherwise the method raises ERichViewError[608] exception. Items are indexed from 0 to ItemCount[144]-1, GetItemStyle[194] returns type of item. Items of subdocuments (table cells[567]) are not included in the items range of the main document; for items in cells, use Cell.GetRVData[575].GetBulletInfo.

Output parameters:

AName – name of *bullet*[96]. It can also be read using GetItemText[195] method.

AImageList – image list (a reference to image list, do not free it).

AImageIndex – index in **AImageList**. It can also be read using GetItemExtraIntPropertyEx[193] method.

ATag – *tag* of the item. Use SetItemTag[231] or SetBulletInfo[227] to change *tag* of item. This value can also be read using GetItemTag[195] method.

Additional item properties are returned by the methods GetItemExtraIntProperty[193] and GetItemExtraStrProperty[193].

Unicode note: names of non-text items are ANSI strings.

See also methods:

- SetBulletInfo[227];

- GetItemStyle 194;
- GetItemExtraIntProperty 193;
- GetItemExtraStrProperty 193.

See also properties:

- ItemCount 144.

See also:

- Obtaining RichView items 50;
- Item types 29.
- *Tags* 34.

1.4.1.1.3.45 TRichView.GetCheckpointByNo

Returns the **No**-th *checkpoint*.

```
function GetCheckpointByNo(No: Integer): TCheckpointData 622;
```

The first *checkpoint* has index = 0.

Note 1: this function contains some calculations inside (iterations through checkpoints, from the 0-th to the **No**-th *checkpoint*).

Note 2: this method does not work with *checkpoints* in subdocuments (table cells 567)

See also methods:

- GetCheckpointNo 186;
- GetItemCheckpoint 191.

See also:

- "Checkpoints" 30.

1.4.1.1.3.46 TRichView.GetCheckpointInfo

Returns information about the *checkpoint* **CheckpointData**

```
procedure GetCheckpointInfo(CheckpointData: TCheckpointData 622;
  var Tag: TRVTag 645; var Name: String;
  var RaiseEvent: Boolean);
```

Output parameters

Tag – *tag* of checkpoint;

Name – name of the *checkpoint*, any string;

RaiseEvent – "raise event" flag; if set, RichView can generate OnCheckpointVisible 238 event for this *checkpoint*.

Unicode note: names of *checkpoints* are Unicode strings for Delphi/C++Builder 2009 or newer, and ANSI strings for older versions of Delphi/C++Builder.

See also properties:

- CPEventKind 139.

See also events:

- OnCheckpointVisible 238.

See also:

- "Checkpoints" 30;
- "Tags" 34.

1.4.1.1.3.47 TRichView .GetCheckpointItemNo

Returns index of the RichView item which owns the checkpoint **CheckpointData**

```
function GetCheckpointItemNo(CheckpointData: TCheckpointData[622]): Integer;
```

Note: this method does not work with checkpoints in subdocuments (table cells[567])

See also:

- Item types[29];
- "Checkpoints"[30].

1.4.1.1.3.48 TRichView .GetCheckpointNo

Returns index of the *checkpoint* **CheckpointData**

```
function GetCheckpointNo(CheckpointData: TCheckpointData[622]): Integer;
```

The first *checkpoint* has index = 0.

Note 1: this function contains some calculations inside (iterations through *checkpoints*, from the 0-th to **CheckpointData**)

Note 2: this method does not work with *checkpoints* in subdocuments (table cells[567])

See also methods:

- GetCheckpointByNo[185];
- GetCheckpointItemNo[186].

See also:

- "Checkpoints"[30].

1.4.1.1.3.49 TRichView .GetCheckpointXY

Returns **X** and **Y** coordinates of the *checkpoint* (relative the top left corner of the document, i.e. top left corner of scrollable area)

```
procedure GetCheckpointXY(CheckpointData: TCheckpointData[622];
  var X,Y: Integer);
```

You can use the returned **Y** to scroll document to this *checkpoint*.

These coordinates are the coordinates of the top left corner of the item which owns this *checkpoint* (for *checkpoint* at the bottom of the document the method returns (X=0, Y=DocumentHeight[142]).

This method must be called only when the document is formatted.

See also methods:

- GetCheckpointY[186];
- GetCheckpointYEx[187];
- ScrollTo[513].

See also:

- "Checkpoints"[30];
- Scrolling[46].

1.4.1.1.3.50 TRichView .GetCheckpointY

Returns vertical coordinate of the **no**-th *checkpoint*

```
function GetCheckpointY(no: Integer): Integer;
```

The first *checkpoint* has index (i.e. **no** parameter) = 0.

The coordinate is relative to the top of document (top of scrollable area)

You can use the returned value to scroll document to this *checkpoint*.

This method must be called only when the document is formatted.

Note: this function contains some calculations inside (iterations through *checkpoints*, from the 0-th to the **no**-th checkpoint).

This function ignores *checkpoints* in table cells.

See also methods:

- GetCheckpointYEx 187;
- GetCheckpointXY 186;
- ScrollTo 513.

See also:

- "Checkpoints" 30;
- Vertical scrolling 46.

1.4.1.1.3.51 TRichView.GetCheckpointYEx

Returns vertical coordinate of the *checkpoint* **CheckpointData**

```
function GetCheckpointYEx(CheckpointData: TCheckpointData 622): Integer;
```

Coordinate is relative to the top of the document (top of scrollable area)

You can use the returned value to scroll document to this *checkpoint*.

This method must be called only when the document is formatted.

See also methods:

- GetCheckpointY 186;
- GetCheckpointXY 186;
- ScrollTo 513.

See also:

- "Checkpoints" 30;
- Vertical scrolling 46.

1.4.1.1.3.52 TRichView.GetControlInfo

Returns main properties of the item of "inserted control" type 94

```
procedure GetControlInfo(ItemNo: Integer;
  var AName: TRVAnsiString 623; var Actrl: TControl;
  var AVAlign: TRVVAlign 648; var ATag: TRVTag 645);
```

Input parameter:

ItemNo – index of the item. The item must be of "inserted control" type 94 (*rvsComponent* 663), otherwise the method raises ERichViewError 608 exception. Items are indexed from 0 to ItemCount 144-1, GetItemStyle 194 returns type of item. Items of subdocuments (table cells 567) are not included in the items range of the main document; for items in cells, use Cell.GetRVData 575.GetControlInfo.

Output parameters:

AName – name of control item 94. Do not confuse with **Actrl**.Name property! This value can also be read using GetItemText 195 method.

Actr – the control itself. This method returns control owned by RichView, do not not destroy it.

AVAlign – vertical alignment of the control.

ATag – *tag* of the item. Do not confuse with **Actrl**.Tag property!. Use SetItemTag 231 or SetControlInfo 228 to

change *tag* of item. This value can also be read using GetItemTag[195] method.

Additional item properties are returned by the methods GetItemExtraIntProperty[193] and GetItemExtraStrProperty[193].

Unicode note: names of non-text items are ANSI strings.

See also methods:
- SetControlInfo[228];
- GetItemStyle[194];
- GetItemExtraIntProperty[193];
- GetItemExtraStrProperty[193].

See also properties:
- ItemCount[144].

See also:
- Obtaining RichView items[50];
- Item types[29];
- "Tags"[34].

1.4.1.1.3.53 TRichView.GetFirstCheckpoint

Returns value identifying the first *checkpoint* in the document, or **nil** if the document has no *checkpoints*.

```
function GetFirstCheckpoint: TCheckpointData[622];
```

GetFirstCheckpoint + GetNextCheckpoint[199] allow to iterate through all *checkpoints* in the document.

Checkpoints from subdocuments (table cells[567]) are not included in *checkpoints* list of the document.

See also methods:
- GetNextCheckpoint[199], contains example;
- GetLastCheckpoint[198];
- GetPrevCheckpoint[202];
- GetCheckpointInfo[185];
- GetCheckpointXY[186].

See also:
- "Checkpoints"[30].

1.4.1.1.3.54 TRichView.GetFocusedItem

Returns the item having keyboard focus.

```
procedure GetFocusedItem(var ARVData: TCustomRVFormattedData[608];
  var AItemNo: Integer);
```

There are two types of items which can have keyboard focus:
- hypertext[35] (text or pictures)
- inserted controls[94].

If hypertext item has focus, pressing Enter invokes OnJump[247] event for the focused link.

This method returns the focused item (the **AItemNo**-th item of **ARVData** object).

ARVData can be TRichView.RVData[151] or table cell[113].

This method must be called only when the document is formatted.

If **ARVData**=**nil**, there is no focused item.

Focused items are supported only in TRichView, not in TRichViewEdit 282.

See also:

- TabNavigation 157 property.

1.4.1.1.3.55 TRichView.GetHotspotInfo

Returns main properties for item of *hotspot* 97 type 97

```
procedure GetHotspotInfo(ItemNo: Integer; var AName: TRVAnsiString 623;
  var AImageIndex, AHotImageIndex: Integer;
  var AImageList: TCustomImageList;
  var ATag: TRVTag 645);
```

Input parameter:

ItemNo – index of the item. The item must be of *hotspot* 97 type (*rvsHotspot* 663), otherwise the method raises ERichViewError 608 exception. Items are indexed from 0 to ItemCount 144-1, GetItemStyle 194 returns type of item. Items of subdocuments (table cells 567) are not included in the items range of the main document; for items in cells, use Cell.GetRVData 575.GetHotspotInfo.

Output parameters:

AName – name of *hotspot* 97. It can also be read using GetItemText 195 method.

AImageList – image list (a reference to image list, do not free it).

AImageIndex – index in **AImageList** for normal image.It can also be read using GetItemExtraIntPropertyEx 193 method.

AHotImageIndex – index in **AImageList** for "hot" image. This image is displayed under the mouse pointer (in TRichView, or in TRichViewEdit in hypertext mode), or when user moves the caret to this item (in TRichViewEdit). It can also be read using GetItemExtraIntPropertyEx 193 method.

ATag – *tag* of the item. Use SetItemTag 231 or SetHotspotInfo 229 to change *tag* of item. This value can also be read using GetItemTag 195 method.

Additional item properties are returned by the methods GetItemExtraIntProperty 193 and GetItemExtraStrProperty 193.

Unicode note: names of non-text items are ANSI strings.

See also methods:

- SetHotspotInfo 229;
- GetItemStyle 194;
- GetItemExtraIntProperty 193;
- GetItemExtraStrProperty 193.

See also properties:

- ItemCount 144.

See also:

- Obtaining RichView items 50;
- Item types 29;
- *Tags* 34.

1.4.1.1.3.56 TRichView .GetItem

Returns object representing one item in document by its index **ItemNo** (zero-based)

```
function GetItem(ItemNo: Integer): TCustomRVItemInfo[526];
```

(introduced in version 1.4)

Items are indexed from 0 to ItemCount[144]-1, GetItemStyle[194] returns type of item. Items of subdocuments (table cells[567]) are not included in the items range of the main document; for items in cells, use Cell.GetRVData[575].GetItem.

Do not free this object yourself.

Usually this method if used to get an item object for tables[98], labels[101], sequences[102], footnotes[104], endnotes[103], references to notes[108].

This method is inverse to GetItemNo[194].

See also methods:

- AddItem[166];
- GetItemStyle[194].

See also methods of TRichViewEdit:

- GetCurrentItem[311];
- GetCurrentItemEx[311].

See also properties:

- ItemCount[144].

See also events:

- OnItemAction[244].

See also:

- Obtaining RichView items[50].

1.4.1.1.3.57 TRichView .GetItemAt

Returns item[86] at the position **X**,**Y**.

```
function GetItemAt(X, Y: Integer; var RVData: TCustomRVFormattedData[608];
  var ItemNo, OffsetInItem: Integer; Strict: Boolean): Boolean;
```

Input parameters:

X,**Y** – coordinates relative to the top level corner of document (scrollable area)

Strict:

- *True* – the method returns an item which is exactly at the specified location;
- *False* – the method returns a position where caret should be moved, if the user would click the mouse at (**X**,**Y**)

Output parameters:

RVData – object containing this item (RichView.RVData[151] or table cell[567] or RVData of cell inplace editor)

ItemNo – index of item (in the items list of RVData), or -1 if no item was found.

OffsetInItem – position of this item:

- for non-text: 0 (closer to the beginning of the item) or 1 (to the end);
- for text: position is before the **OffsetInItem**-th character of text; characters are indexed from 1, the last position is Length(text)+1.

Return value:

"item found?"

This method must be called only when the document is formatted.

See also:

- GetWordAt 205.

See also events:

- OnRVMouseDown 253 (contains example);
- OnRVMouseUp 254.

Demo projects:

- **Demos\Delphi\Assorted\PlanetQuiz**
- **Demos\CBuilder\Assorted\PlanetQuiz**

1.4.1.1.3.58 TRichView.GetItemCheckpoint

Returns *checkpoint* associated with the **ItemNo**-th item, or **nil** if this item does not have associated *checkpoint*.

```
function GetItemCheckpoint(ItemNo: Integer): TCheckpointData 622;
```

Items are indexed from 0 to ItemCount 144-1. Items of subdocuments (table cells 567) are not included in the items range of the main document; for items in cells, use Cell.GetRVData 575.GetItemCheckpoint.

See also methods:

- GetCheckpointInfo 185;
- GetCheckpointXY 186.

See also:

- "Checkpoints" 30.

1.4.1.1.3.59 TRichView.GetItemCoords

The methods return coordinates of the item in RichView document.

```
function GetItemCoords(ItemNo: Integer;
  var Left,Top: Integer): Boolean;
function GetItemClientCoords(ItemNo: Integer;
  var Left,Top: Integer): Boolean;
```

(introduced in version 1.4)

Input parameter:

ItemNo – index of the item. Items are indexed from 0 to ItemCount 144-1. Items of subdocuments (table cells 567) are not included in the items range of the main document.

Output parameters:

Left,**Top** – coordinates of the top left corner of the item, pixels.

GetItemCoords returns document coordinates, i.e. (0,0) is in the top left corner of the scrollable area.

GetItemClientCoords returns client coordinates, i.e. (0,0) is in the top left corner of the control window.

These methods must be called only when the document is formatted.

Limitations:

- the methods cannot be used to get coordinates of items inside table cells;

- the methods return only the top left coordinates;
- for text items occupying several lines, the methods return the coordinates for the first line;
- for left- and right-aligned objects, the methods return the position of their placeholder in a text, not the position of a floating object.

GetItemCoordsEx 192 does not have these limitations.

Return value:

True if coordinates were successfully retrieved (document must be formatted)

See also:

- ClientToDocument, DocumentToClient 176.

See also methods of TCustomRVFormattedData

- GetOriginEx 611.

1.4.1.1.3.60 TRichView.GetItemCoordsEx

Returns coordinates of the item in RichView document.

```
function GetItemCoordsEx(RVData: TCustomRVFormattedData608; ItemNo, Part: Integer;
  AllowFloating: Boolean; var R: TRect): Boolean;
```

(introduced in version 14)

Input parameters:

RVData, ItemNo define the position of the item. **RVData** – an object containing this item (RichView.RVData 151 or table cell 567 or RVData of cell inplace editor). **ItemNo** – an index of the item (in the items list of **RVData**).

Part – the item part. For non-text items and text items occupying a single line, it should be 0. For text items occupying more than one line, **Part** must be 0 for the first line, 1 for the second line and so on (you can call this method until it returns *False* for a non-existing part).

AllowFloating specifies which coordinates should be returned for left- and right-aligned 648 objects. If *True*, coordinates of a floating object are returned. If *False*, coordinates of a placeholder in a text are returned.

Output parameters:

R – coordinates of the item (or item part), in pixels. These are document coordinates, i.e. (0,0) is in the top left corner of the scrollable area. If you need client coordinates, convert **R**.TopLeft and **R**.BottomRight using DocumentToClient 176.

The method must be called only when the document is formatted.

This method takes cell rotation 573 into account.

Return value:

True if coordinates were successfully retrieved (document must be formatted; and **Part** must be valid).

See also:

- GetItemCoords 191.

1.4.1.1.3.61 TRichView.GetItemExtraIntProperty -Ex

The methods return a value of the specified integer property for the **ItemNo**-th item.

```
function GetItemExtraIntProperty(ItemNo: Integer;
  Prop: TRVExtraItemProperty; var Value: Integer): Boolean;
function GetItemExtraIntPropertyEx(ItemNo, Prop: Integer;
  var Value: Integer): Boolean;
```

(introduced in versions 1.7 and 15)

ItemNo – index of the item. Items are indexed from 0 to ItemCount-1, GetItemStyle returns type of item. Items of subdocuments (table cells) are not included in the items range of the main document; for items in cells, use Cell.GetRVData.GetItemExtraIntProperty[Ex].

Value receives value of the property identified by **Prop**.

In **GetItemExtraIntProperty**, **Prop**'s type is TRVExtraItemProperty. See information about this type for the list of properties.

In **GetItemExtraIntPropertyEx**, **Prop**'s type is Integer. If **Prop** can be converted to TRVExtraItemProperty, **GetItemExtraIntPropertyEx** works like **GetItemExtraIntProperty**. In addition, it supports properties identified by rveipc*** constants

Return value:

True, if this item has this property. *False*, if not.

See also:

- SetItemExtraIntProperty[Ex];
- GetItemExtraStrProperty.

See also methods of TRichViewEdit:

- GetCurrentItemExtraIntProperty[Ex].

1.4.1.1.3.62 TRichView.GetItemExtraStrProperty -Ex

The methods return a value of the specified string property for the **ItemNo**-th item

```
function GetItemExtraStrProperty(ItemNo: Integer;
  Prop: TRVExtraItemStrProperty; var Value: String): Boolean;
function GetItemExtraStrPropertyEx(ItemNo, Prop: Integer;
  var Value: String): Boolean;
```

(introduced in versions 1.9 and 15)

ItemNo – index of the item. Items are indexed from 0 to ItemCount-1, GetItemStyle returns type of item. Items of subdocuments (table cells) are not included in the items range of the main document; for items in cells, use Cell.GetRVData.GetItemExtraStrProperty[Ex].

Value receives value of the property identified by **Prop**.

In **GetItemExtraStrProperty**, **Prop**'s type is TRVExtraItemStrProperty. See information about this type for the list of properties.

In **GetItemExtraStrPropertyEx**, **Prop**'s type is Integer. If **Prop** can be converted to TRVExtraItemStrProperty, **GetItemExtraStrPropertyEx** works like **GetItemExtraStrProperty**. In addition, it supports properties identified by rvespc*** constants

Return value:

True, if this item has this property. *False*, if not.

Unicode note: string properties are Unicode strings for Delphi/C++Builder 2009 or newer, and ANSI strings for older versions of Delphi/C++Builder.

See also:

- SetItemExtraStrProperty[Ex] 231;
- GetItemExtra 193 Int 193 Property[Ex] 193.

See also methods of TRichViewEdit:

- GetCurrentItemExtraStrProperty[Ex] 313.

1.4.1.1.3.63 TRichView.GetItemNo

Returns index of item (zero-based).

```
function GetItemNo(Item: TCustomRVItemInfo 526): Integer;
```

(introduced in version 1.4)

Item is an object representing the item. Usually this method if used to get index if table item 528.

This method is inverse to GetItem 190.

Returned value can be used, for example, in methods:

TRichViewEdit.BeginItemModify 301/EndItemModify 307

See also methods:

AddItem 166.

1.4.1.1.3.64 TRichView.GetItemPara

Returns the index of paragraph style for paragraph containing the **ItemNo**-th item.

```
function GetItemPara(ItemNo: Integer): Integer;
```

(Introduced in version 1.3)

Parameter:

ItemNo – index of the item. Items are indexed from 0 to ItemCount 144-1, GetItemStyle 194 returns type of item. Items of subdocuments (table cells 567) are not included in the items range of the main document; for items in cells, use Cell.GetRVData 575.GetItemPara.

Return value:

The returned value is an index in the collection of Style 156.ParaStyles 398. The method never returns -1, even if this item is not the first paragraph item. The method returns the same value for all items in the same paragraph.

See also:

- IsParaStart 208;
- IsFromNewLine 207.

See also:

- Paragraphs 37.

1.4.1.1.3.65 TRichView.GetItemStyle

Returns type (style) of the **ItemNo**-th item.

```
function GetItemStyle(ItemNo: Integer): Integer;
```

Parameters:

ItemNo – index of item (from 0 to ItemCount 144-1). Items are indexed from 0 to ItemCount 144-1. Items of subdocuments (table cells 567) are not included in the items range of the main document; for items in cells, use

Cell.GetRVData[575].GetItemStyle.

Return value:

If the item is a text item[88], this function returns zero or positive value: index of text style of this item in the collection of styles (Style[156].TextStyles[404]).

If the item is not a text item, this function returns type of this item. It is a negative value, see rvs**** constants[663].

See also properties of TRichViewEdit:

- CurItemStyle[289].

See also:

- Item types[29].

1.4.1.1.3.66 TRichView.GetItemTag

Returns *tag* of the **ItemNo**-th item

```
function GetItemTag(ItemNo: Integer): TRVTag[645];
```

Parameter:

ItemNo – index of the item. Items are indexed from 0 to ItemCount[144]-1. Items of subdocuments (table cells[567]) are not included in the items range of the main document; for items in cells, use Cell.GetRVData[575].GetItemTag.

Use SetItemTag[231] to change tag of item.

See also methods:

- SetItemTag[231].

See also method of TRichViewEdit:

- GetCurrentTag[316].

See also:

- RichView items' "tags"[34].

1.4.1.1.3.67 TRichView.GetItemText -A -W

Returns a text of text item, or a name of non-text item.

```
function GetItemText(ItemNo: Integer): String;
function GetItemTextA(ItemNo: Integer): TRVAnsiString[623];
function GetItemTextW(ItemNo: Integer): TRVUnicodeString[647];
```

(Introduced in version 1.3, 1.7)

ItemNo – index of the item. Items of subdocuments (table cells[567]) are not included in the items range of the main document; for items in cells, use Cell.GetRVData[575].GetItemText*.

Unicode notes:

GetItemTextA always returns ANSI string:

- If the item is a non-text item, or a text item of ANSI style, the text is returned as it is.
- If the item is a Unicode text item, the function converts Unicode text to ANSI (Unicode text is converted basing on Charset[434] of the item text style).

GetItemTextW always returns Unicode string:

- If the item is a Unicode text item, the text is returned as it is.
- If the item is a non-text item, or a text item of ANSI style, the function converts ANSI text to Unicode (ANSI text is converted basing on Charset[434] of the item text style, or on RVStyle.DefCodePage[388] for non-text

items).

GetItemText works like **GetItemTextW** in Delphi/C++Builder 2009 or newer, and like **GetItemTextA** in the older versions of Delphi/C++Builder.

(There is one more version of this method, GetItemTextR, returning TRVRawByteString 638 value. This method returns text in a way it is stored in TRichView)

See also:

- SetItemText -A -W 232.

See also method of TRichViewEdit:

- GetCurrentItemText -A -W 313.

See also:

- Obtaining Items of RichView 50;
- Unicode in RichView 64.

1.4.1.1.3.68 TRichView .GetItemVAlign

Returns vertical alignment (position relative to the line) of the **ItemNo**-th item.

```
function GetItemVAlign(ItemNo: Integer): TRVVAlign 648;
```

(introduced in version 12)

Parameter:

ItemNo – index of the item. Items are indexed from 0 to ItemCount 144-1. Items of subdocuments (table cells 567) are not included in the items range of the main document; for items in cells, use Cell.GetRVData 575 .GetItemVAlign.

This method must not be called for text items 88, tables 98, *breaks* 93, tabs 89, list markers 99, footnotes 104 and endnotes 103.

Use SetItemVAlign 233 to change vertical alignment.

See also methods:

- SetItemVAlign 233.

See also method of TRichViewEdit:

- GetCurrentItemVAlign 314

1.4.1.1.3.69 TRichView .GetJumpPointItemNo

Returns index of hypertext item (*hotspot* 97, *hot-picture* 93 or text hypertext) by its hypertext **id**.

```
function GetJumpPointItemNo(id: Integer): Integer;
```

Warning: do not use this function for document containing tables 111. When the link is in table cell, or in cell inplace-editor, it can not be addressed using only an index of item. Use GetJumpPointLocation 197 instead.

Parameter

id – identifier of the hypertext link; links are numbered sequentially starting from FirstJumpNo 144.

Return value

Index of the hypertext item (in range 0..ItemCount 144-1). If there is no link with such **id**, the method returns -1.

Note: this function contains some calculations inside (iterations through hyperlinks, from the first link to the requested one)

For TCustomRichViewEdit[282], if ReadOnly[294]=False, this function always returns -1 (except for calling it from inside OnWriteHyperlink[267], OnJump[247] or OnRVMouseMove[254]).

See also:

- RichView hypertext[35].

1.4.1.1.3.70 TRichView.GetJumpPointLocation

Returns information required to access item by its hypertext **id**.

```
procedure GetJumpPointLocation(id: Integer;
  var RVData: TCustomRVFormattedData[608]; var ItemNo: Integer);
```

(introduced in version 1.4)

Input parameter

id – identifier of the hypertext link; links are numbered sequentially starting from FirstJumpNo[144].

Output parameters

RVData and **ItemNo** define location of the hypertext item with the given hypertext **id**.

Hypertext item is the **ItemNo**-th item in **RVData** object.

RVData is a RichView document. It can be RichView.RVData[151], table cell[567], or RVData of cell inplace-editor.

If there is no jump with such **id**, the method returns -1 in **ItemNo**.

Note: this function contains some calculations inside (iterations through hyperlinks, from the first link to the requested one)

For TCustomRichViewEdit[282], if ReadOnly[294]=False, this function always returns -1 (except for calling it from inside OnWriteHyperlink[267], OnJump[247] or OnRVMouseMove[254]).

See also:

- RichView hypertext[35].

1.4.1.1.3.71 TRichView.GetJumpPointY

Returns vertical coordinate of the hypertext link (*hotspot*[97], *hot-picture*[93] or text hypertext), identified by **id**

```
function GetJumpPointY(id: Integer): Integer;
```

Parameter

id – identifier of the hypertext link; links are numbered sequentially starting from FirstJumpNo[144].

Return value

You can use the returned value to scroll document to the position of this link (useful when implementing "Back" button functionality)

If there is no link with such **id**, the method returns 0 (top of the document)

This method must be called only when the document is formatted.

Note: this function contains some calculations inside (iterations through jumps, from the first jump to requested one)

For TCustomRichViewEdit[282], if ReadOnly[294]=False, this function always returns 0 (except for calling it from

inside OnJump[247] or OnRVMouseMove[254]).

See also methods:

- ScrollTo[513]

See also:

- RichView Hypertext[35]

1.4.1.1.3.72 TRichView .GetLastCheckpoint

Returns value identifying the last *checkpoint* in the document, or **nil** if the document has no *checkpoints*.

```
function GetLastCheckpoint: TCheckpointData[622];
```

(introduced in version 1.5)

GetLastCheckpoint + GetPrevCheckpoint[202] allow to iterate through all *checkpoints* in the document.

Checkpoints from subdocuments (table cells) are not included in the *checkpoints* list of document.

See also methods:

- GetNextCheckpoint[199], contains example;
- GetFirstCheckpoint[188];
- GetPrevCheckpoint[202];
- GetCheckpointInfo[185];
- GetCheckpointXY[186].

See also:

- "Checkpoints"[30].

1.4.1.1.3.73 TRichView .GetLineNo

Returns line of the position specified by **ItemNo** and **ItemOffs**.

```
function GetLineNo(ItemNo, ItemOffs: Integer): Integer;
```

(introduced in version 1.7)

Parameters

ItemNo – index of the item, in range from 0 to ItemCount[144]-1.

ItemOffs – offset in item.

For non-text items, **ItemOffs**=0 specifies position before the item, **ItemOffs**=1 specifies position after the item (this is not important for this function, because non-text item cannot be placed on several lines).

For text items, ItemOffs specifies position before the **ItemOffs**-th character:

- **ItemOffs**=1 – position before the text item (before the first character);
- **ItemOffs**=2 – position before the second character;
- ....
- **ItemOffs**=GetOffsAfterItem[200](ItemNo) – position after the last character of the text item.

Return value

Line index. The first line has index 1. Number of lines depends on word-wrapping.

Example:

```
with MyRichView do
  r := GetLineNo(ItemCount[144]-1, GetOffsAfterItem[200](ItemCount[144]-1))
```

r receives number of lines in MyRichView.

Warning: this function contains calculations inside, and it may take a while to execute it. (There is an idea how to make it lightning-fast in future, but it will be implemented later).

This method must be called only when the document is formatted.

See also:

- GetItemStyle 194.

See also methods of TCustomRichViewEdit:

- GetCurrentLineCol 314.

1.4.1.1.3.74 TRichView.GetListMarkerInfo

Returns main properties of list marker 99 (paragraph bullet or numbering)

```
function GetListMarkerInfo(AItemNo: Integer;
  var AListNo, AListLevel, AStartFrom: Integer;
  var AUseStartFrom: Boolean): Integer;
```

(introduced in version 1.7)

Input parameter

AItemNo – index of the item, in range from 0 to ItemCount 144-1. This is not necessary the index of list marker, it can be index of any item in the given paragraph. Items of subdocuments (table cells 567) are not included in the items range of the main document; for items in cells, use Cell.GetRVData 575.GetListMarkerInfo.

Output parameters (valid only if the function returned non-negative value)

AListNo – style of this list maker. This is an index in the collection of list styles (Style 156.ListStyles 396).

AListLevel – list level. This is an index in the collection of list levels (Style.ListStyles[AListNo].Levels 459).

AStartFrom: if **AUseStartFrom**=*True*, and this is a numbered list, this parameter is a value of list counter for this marker. If **AUseStartFrom**=*False*, numbering is continued.

Return value:

If this paragraph does not have a marker, this function returns -1; otherwise it returns item index of the marker.

See also:

- SetListMarkerInfo 233.

1.4.1.1.3.75 TRichView.GetNextCheckpoint

Returns value identifying the next *checkpoint* after the given **CheckpointData**, or **nil** if **CheckpointData** is the last *checkpoint* in the document.

```
function GetNextCheckpoint(CheckpointData: TCheckpointData 622): TCheckpointData 622;
```

GetFirstCheckpoint 188 + **GetNextCheckpoint** allow to iterate through all *checkpoints* in the document.

Checkpoints from subdocuments (table cells 567) are not included in *checkpoints* list of the document.

Example:

```
var Tag: TRVTag 645;
    Name: String;
    CheckpointData: TCheckpointData;
    RaiseEvent: Boolean;
```

```
    s: String;
begin
  MyListBox.Items.Clear;
  CheckpointData := MyRichView.GetFirstCheckpoint[188];
  while CheckpointData<>nil do
  begin
    MyRichView.GetCheckpointInfo[185](CheckpointData, Tag, Name, RaiseEvent);
    s := Format('Name:"%s" Tag:"%s"', [Name, Tag]);
    MyListBox.Items.Add(s);
    CheckpointData := MyRichView.GetNextCheckpoint(CheckpointData);
  end;
end;
```

This code fills listbox MyListBox with information about *checkpoints* of MyRichView.

See also methods:

- GetFirstCheckpoint[188];
- GetLastCheckpoint[198];
- GetPrevCheckpoint[202];
- GetCheckpointInfo[185];
- GetCheckpointXY[186].

See also:

- "Checkpoints"[30].

1.4.1.1.3.76 TRichView .GetOffsAfterItem

Returns offset after the **ItemNo**-th item in the document

```
function GetOffsAfterItem(ItemNo: Integer): Integer;
```

This method is useful for selecting part of document for copying in the Clipboard, or for moving caret to the specified position in editor.

ItemNo – index of the item. Items are indexed from 0 to ItemCount[144]-1, GetItemStyle[194] returns type of item. Items of subdocuments (table cells[567]) are not included in the items range of the main document; for items in cells, use Cell.GetRVData[575].GetOffsAfterItem.

For text items, the method returns length of text + 1 For non-text items, it returns 1.

Example for TRichViewEdit[282]: moving caret to the end of document

```
var ItemNo, Offs: Integer;

  ItemNo := MyRichViewEdit.ItemCount-1;
  Offs := MyRichViewEdit.GetOffsAfterItem(ItemNo);
  MyRichViewEdit.SetSelectionBounds[235](ItemNo, Offs, ItemNo, Offs);
  MyRichViewEdit.Invalidate;
```

See also methods of TRichView:

- SetSelectionBounds[235];
- GetOffsBeforeItem[201].

See also properties of TRichView:

- ItemCount[144].

See also properties of TRichViewEdit:

- CurItemNo[288];
- OffsetInCurItem[294].

See also:

- Working with selection 47;
- How to: move a caret to the beginning or to the end of document in TRichViewEdit 668.

1.4.1.1.3.77 TRichView.GetOffsBeforeItem

Returns offset before the **ItemNo**-th item in the document

```
function GetOffsBeforeItem(ItemNo: Integer): Integer;
```

This method is useful for selecting a part of document for copying in the Clipboard, or for moving caret to the specified position in editor.

ItemNo – index of the item. Items are indexed from 0 to ItemCount 144-1, GetItemStyle 194 returns type of item. Items of subdocuments (table cells 567) are not included in the items range of the main document; for items in cells, use Cell.GetRVData 575.GetOffsBeforeItem.

For text items, the method returns 1 For non-text items, it returns 0.

Example for TRichViewEdit 282: moving caret to the beginning of document

```
var ItemNo, Offs: Integer;

  ItemNo := 0;
  Offs := MyRichViewEdit.GetOffsBeforeItem(ItemNo);
  MyRichViewEdit.SetSelectionBounds 235(ItemNo, Offs, ItemNo, Offs);
  MyRichViewEdit.Invalidate;
```

See also methods of TRichView:

- SetSelectionBounds 235;
- GetOffsAfterItem 200.

See also properties of TRichView:

- ItemCount 144.

See also properties of TRichViewEdit:

- CurItemNo 288;
- OffsetInCurItem 294.

See also:

- Working with selection 47;
- How to: move a caret to the beginning or to the end of document in TRichViewEdit 668;
- Example how to move caret to the beginning of paragraph 371.

1.4.1.1.3.78 TRichView.GetPictureInfo

Returns main properties for item of picture 90 or *hot-picture* 93 type

```
procedure GetPictureInfo(ItemNo: Integer; var AName: TRVAnsiString 623;
  var Agr: TGraphic; var AVAlign: TRVVAlign 648;
  var ATag: TRVTag 645);
```

Input parameter:

ItemNo – index of the item. The item must be of picture 90 or *hot-picture* 93 type (*rvsPicture* 663), otherwise the method raises ERichViewError 608 exception. Items are indexed from 0 to ItemCount 144-1, GetItemStyle 194 returns type of item. Items of subdocuments (table cells 567) are not included in the items range of the main document; for items in cells, use Cell.GetRVData 575.GetPictureInfo.

Output parameters:

AName – name of the item. This value can also be read using GetItemText 195 method.

Agr – graphic object. This method returns object owned by RichView, do not not destroy it.

AVAlign – vertical alignment of the picture.

ATag – *tag* of the item. Use SetItemTag 231 or SetPictureInfo 234 to change *tag* of item. This value can also be read using GetItemTag 195 method.

Additional item properties (including stretching, transparency for bitmaps, link to file name, alternative text for saving to HTML) are returned by the methods GetItemExtraIntProperty 193 and GetItemExtraStrProperty 193.

Unicode note: names of non-text items are ANSI strings.

See also methods:

- SetPictureInfo 234;
- GetItemStyle 194;
- GetItemExtraIntProperty 193;
- GetItemExtraStrProperty 193.

See also properties:

- ItemCount 144.

See also:

- Obtaining RichView items 50;
- Item types 29;
- "Tags" 34.

1.4.1.1.3.79 TRichView.GetPrevCheckpoint

Returns value identifying the previous *checkpoint* before the given **CheckpointData**, or **nil** if **CheckpointData** is the first *checkpoint* in the document.

```
function GetPrevCheckpoint(CheckpointData: TCheckpointData): TCheckpointData;
```

(introduced in version 1.5)

GetLastCheckpoint 198 + **GetPrevCheckpoint** allow to iterate through all *checkpoints* in the document.

Checkpoints from subdocuments (table cells 567) are not included in *checkpoints* list of document.

See also methods:

- GetFirstCheckpoint 188;
- GetLastCheckpoint 198;
- GetNextCheckpoint 199;
- GetCheckpointInfo 185;
- GetCheckpointXY 186.

See also:

- "Checkpoints" 30.

1.4.1.1.3.80 TRichView.GetSelectedImage

Returns the selected image

```
function GetSelectedImage: TGraphic;
```

This method returns image, if the selection consist only of one image. If there is nothing selected, or not only image is selected, this method returns **nil**.

The method returns image owned by RichView, not a copy of it. So do not destroy this image.

This method must be called only when the document is formatted.

See also:

- Working with selection 47.

1.4.1.1.3.81 TRichView.GetSelectionBounds

Returns bounds of selected part of document.

```
procedure GetSelectionBounds(var StartItemNo, StartItemOffs,
  EndItemNo, EndItemOffs: Integer; Normalize: Boolean);
```

Parameters

StartItemNo – index of the first selected item.

StartItemOffs:

- if the first item is a text item 88, then the selection start is before the **StartItemOffs**-th character of string (characters in strings are counted from 1, the last position (after the text item) is text length+1).
- if the first item is not a text, then
  - if **StartItemOffs**=0, then the selection start is before the first item;
  - if **StartItemOffs**=1, then the selection start is after the first item.

EndItemNo – index of the last selected item.

EndItemOffs:

- if the last item is a text item, then the selection end is before the **EndItemOffs**-th character of string (characters in strings are counted from 1, the last position (after the text item) is text length+1).
- if the last item is not a text, then
  - if **EndItemOffs**=0, then the selection end is before the last item
  - if **EndItemOffs**=1, then the selection end is after the last item.

Normalize:

- if *True*, the upper bound of the selection is returned in **StartItemNo**:**StartItemOffs**, and the lower bound is in **EndItemNo**:**EndItemOffs**;
- if *False*, bounds are returned as they were selected by user.

The selection is empty if

- **StartItemNo**=-1 or
- **StartItemNo**=**EndItemNo** and **StartItemOffs**=**EndItemOffs**.

This method must be called only when the document is formatted.

If you want to get selection bounds using richedit-like parameters (SelStart and SelLength), use RVGetSelection from RVLinear 621 unit instead.

See also methods:

- SetSelectionBounds 235.

See also:

- Working with selection. 47

1.4.1.1.3.82 TRichView .GetSelText

These methods return the selected part of the document as a text.

```
function GetSelText: String;
function GetSelTextA: TRVAnsiString 623;
function GetSelTextW: TRVUnicodeString 647;
```

If there is nothing selected, these methods return '' (empty string)

Unicode notes:

GetSelTextA always returns ANSI string:

- If the item is a non-text item, or a text item of ANSI style, the text is returned as it is.
- If the item is a Unicode text item, the function converts Unicode text to ANSI (Unicode text is converted basing on Charset 434 of the item text style).

GetSelTextW always returns Unicode string:

- If the item is a Unicode text item, the text is returned as it is.
- If the item is a non-text item, or a text item of ANSI style, the function converts ANSI text to Unicode (ANSI text is converted basing on Charset 434 of the item text style, or on RVStyle.DefCodePage 388 for non-text items).

GetSelText works like **GetSelTextW** in Delphi/C++Builder 2009 or newer, and like **GetSelTextA** in the older versions of Delphi/C++Builder.

These methods must be called only when the document is formatted.

See also:

- Working with selection 47.

1.4.1.1.3.83 TRichView .GetTextInfo

Returns text and *tag* for the **ItemNo**-th item

```
procedure GetTextInfo(ItemNo: Integer; var AText: String;
  var ATag: TRVTag 645);
```

Input parameter:

ItemNo – index of the item. The item must be of text type (GetItemStyle 194 must return zero or positive value for this item), otherwise the method raises ERichViewError 608 exception. Items are indexed from 0 to ItemCount 144 -1. Items of subdocuments (table cells 567) are not included in the items range of the main document; for items in cells, use Cell.GetRVData 575.GetTextInfo.

Output parameters:

AText – item text (visible text). This is the same value as returned by GetItemText 195.

ATag – *tag* of the item. Use SetItemTag 231 to change *tag* of item. The *tag* can also be returned by GetItemTag 195.

See also methods:

- GetItemStyle 194;
- GetItemText 195, SetItemText 232;
- GetItemTag 195, SetItemTag 231.

See also properties:

- ItemCount 144.

See also:

- Obtaining items 50;
- Item types 29;
- "Tags" 34.

1.4.1.1.3.84 TRichView .GetWordAt -A -W

Returns word at the specified coordinates

```
procedure GetWordAt(X,Y: Integer;
  var RVData: TCustomRVFormattedData608; var ItemNo: Integer;
  var Word: String); overload;
function GetWordAtA(X,Y: Integer): TRVAnsiString623;
function GetWordAtW(X,Y: Integer): TRVUnicodeString647
function GetWordAt(X,Y: Integer): String;
```

Input parameters

(**X**,**Y**) – client coordinates (coordinates relative to the top left corner of RichView window).

Output parameters

Word (or returned value for the simplified versions) receives word (if it is a text item 88), or name of non-text item. See the Unicode note below. "Word" is defined as a part of string between delimiters. Delimiters are listed in Delimiters 140 property.

ItemNo receives index of item at (**X**,**Y**), or -1 if there is no item at this point. This is an index of item in **RVData** object (it may be the main document, cell, or cell inplace editor).

Unicode notes: *(since TRichView version 11)* the advanced version of **GetWordAt** returns word as a String parameter (Unicode for Delphi/C++Builder 2009 or newer, ANSI for older versions of Delphi/C++Builder). A conversion is performed from Unicode/ANSI text items to String, if necessary. See the notes below for the simplified versions for the details of this conversion.

GetWordAtA always returns ANSI string:

- If the item is a non-text item, or a text item of ANSI style, the text is returned as it is.
- If the item is a Unicode text item, the function converts Unicode text to ANSI (Unicode text is converted basing on Charset 434 of the item text style).

GetWordAtW always returns Unicode string:

- If the item is a Unicode text item, the text is returned as it is.
- If the item is a non-text item, or a text item of ANSI style, the function converts ANSI text to Unicode (ANSI text is converted basing on Charset 434 of the item text style, or on RVStyle.DefCodePage 388 for non-text items).

The simplified version of **GetWordAt** works like **GetWordAtW** in Delphi/C++Builder 2009 or newer, and as **GetWordAtA** in the older versions of Delphi/C++Builder.

For example, you can use this method inside OnRVMouseDown 253 and OnRVMouseUp 254.

This method must be called only when the document is formatted.

Example: displaying the clicked words (only text items!) in Label1.

We cannot use simplified version of **GetWordAt** because we cannot know if the returned text is for text item.

```
uses CRVFData;
...
procedure TForm1.RichView1RVMouseUp(Sender: TCustomRichView;
  Button: TMouseButton; Shift: TShiftState; ItemNo, X, Y: Integer);
var
```

```
  LItemNo: Integer;
  LRVData: TCustomRVFormattedData;
  LWord: String;
begin
  ItemNo<0 then
    exit; // we already know that the mouse is not above item
  Sender.GetWordAt(X,Y, LRVData, LItemNo, LWord);
  if LItemNo<0 then
    exit;
  if LRVData.GetItemStyle(194)(LItemNo)>=0 then // text item?
    Label1.Caption := LWord;
end;
```

There is one more version of this method: GetWordAtR. It returns text in Word as it is, in TRVRawByteString(638) parameter.

See also events:

- OnRVMouseDown(253);
- OnRVMouseUp(254);
- OnRVRightClick(255);
- OnRVDblClick(250).

See also properties:

- Delimiters(140).

See also methods:

- SelectWordAt(224)

1.4.1.1.3.85 TRichView.InsertRVFFromStream

Inserts RVF content from **Stream** in the position of the document specified as **Index** (the first inserted item will have this index)

```
function InsertRVFFromStream(Stream: TStream; Index: Integer): Boolean;
```

This is the only method of RichView (not RichViewEdit) inserting items in any position.

Index – index of the item, must be in range from 0 to ItemCount(144). Items of subdocuments (table cells(567)) are not included in the items range of the main document.

Method type: viewer-style.

Setting for RVF loading can be changed in the TRichView component editor(134).

If style templates are used(158), and RVFTextStylesReadMode(154)=RVFParaStylesReadMode(154) =*rvf_sInsertMerge*, and StyleTemplateInsertMode(156)<>*rvstimIgnoreSourceStyleTemplates*, the method merges style templates from RVF into Style(156).StyleTemplates(402), and reads text and paragraph styles according to StyleTemplateInsertMode(156). The method calls OnStyleTemplatesChange(266) event.

Return value: "Was reading successful?"

Example

This example shows how to combine several RVF documents in one.

```
procedure AddDoc(const FileName: String; rv: TCustomRichView);
var Stream: TFileStream;
begin
  Stream := TFileStream.Create(FileName, fmOpenRead);
  rv.InsertRVFFromStream(Stream, rv.ItemCount(144));
  Stream.Free;
end;
  ...
```

```
  RichView1.Clear[175];
  RichView1.DeleteUnusedStyles[181](True, True, True);
  AddDoc('c:\file1.rvf', RichView1);
  AddDoc('c:\file2.rvf', RichView1);
  AddDoc('c:\file3.rvf', RichView1);
  RichView1.Format[182];
```

If you want to add page breaks between documents, use the following code

```
procedure AddDoc(const FileName: String; rv: TCustomRichView);
var Stream: TFileStream;
  ItemCount: Integer;
begin
  ItemCount := rv.ItemCount[144];
  Stream := TFileStream.Create(FileName, fmOpenRead);
  rv.InsertRVFFromStream(Stream, rv.ItemCount[144]);
  Stream.Free;
  if (ItemCount>0) and (ItemCount<rv.ItemCount[144]) then
    rv.PageBreaksBeforeItems[149][ItemCount] := True;
end;
```

See also properties:

- RVFOptions[152];
- RVFWarnings[155];
- RVFTextStylesReadMode[154] (this method type: RVF insertion);
- RVFParaStylesReadMode[154] (this method type: RVF insertion).

See also events:

- OnRVFImageListNeeded[252];
- OnRVFControlNeeded[251];
- OnRVFPictureNeeded[252];
- OnControlAction[239] (ControlAction=*rvcaAfterRVFLoad*);
- OnStyleTemplatesChange[266].

See also methods:

- AppendRVFFromStream[173];
- LoadRVFFromStream[211];
- LoadRVF[211];
- Format[182];
- FormatTail[183].

See also methods of TRichViewEdit:

- InsertRVFFromStreamEd[325].

- **See also:**

- TRichView component editor[134];
- Saving and loading[58];
- RVF overview[59].

1.4.1.1.3.86 TRichView .IsFromNew Line

Returns "does the **ItemNo**-th item start a new line"?

```
function IsFromNewLine(ItemNo: Integer): Boolean;
```

ItemNo – index of the item. Items are indexed from 0 to ItemCount[144]-1, GetItemStyle[194] returns type of item. Items of subdocuments (table cells[567]) are not included in the items range of the main document; for items in cells, use Cell.GetRVData[575].IsFromNewLine.

This method returns *True* both in cases when this item starts a new paragraph (see IsParaStart[208]) or starts a new line inside the paragraph (it was moved to the new line in editor using Shift + Enter, or added in the mode

SetAddParagraphMode[225](*False*)).

See also methods:

- IsParaStart[208];
- GetItemPara[194].

See also:

- Paragraphs[37].

1.4.1.1.3.87 TRichView .IsParaStart

Returns "is the **ItemNo**-th item a first paragraph item"?

```
function IsParaStart(ItemNo: Integer): Boolean;
```

ItemNo – index of the item. Items are indexed from 0 to ItemCount[144]-1, GetItemStyle[194] returns type of item. Items of subdocuments (table cells[567]) are not included in the items range of the main document; for items in cells, use Cell.GetRVData[575].IsParaStart.

This method returns *False* if this item starts a new line inside the paragraph (i.e. it was moved to the new line in editor using Shift+Enter, or added in the mode SetAddParagraphMode[225](*False*)).

See also methods:

- IsFromNewLine[207];
- GetItemPara[194].

See also:

- Paragraphs[37];
- Example how to move caret to the beginning of paragraph[371].

1.4.1.1.3.88 TRichView .LiveSpellingValidateWord

Removes live spelling underlines from all occurrences of **AWord**.

```
procedure LiveSpellingValidateWord(const AWord: String);
```

(introduced in version 1.9)

Call this method if you added this word in a dictionary or ignore list.

This procedure does not affect further processing of this word. User dictionaries and ignore list must be maintained by spelling checkers.

Unicode note: live spelling interface is Unicode for Delphi/C++Builder 2009 or newer, and ANSI for the older versions of Delphi/C++Builder.

See also:

- Live spelling check in RichView[69].

1.4.1.1.3.89 TRichView .LoadFromStream

Loads document from **Stream**, autodetecting its format (RVF[59]/RTF/text)

```
type
  TRVYesNoAuto = ( rvynaNo, rvynaYes, rvynaAuto );
function LoadFromStream(Stream: TStream; IsTextUnicode: TRVYesNoAuto): Boolean;
```

(introduced in version 10)

This method can detect both normal RTF, and RTF converted to Unicode string.

The method does not load text if it contains characters having codes less than ord(' ').

IsTextUnicode specifies whether the text is Unicode[64]:

- *rvynaNo* – reading text as ANSI;
- *rvynaYes* – reading text as Unicode;
- *rvynaAuto* – autodetecting ANSI/Unicode (wrong detection is very possible).

Method type: viewer-style.

Return value:

"Was reading successful?"

See also methods of TRichView:

- LoadRVFFromStream[211];
- LoadRTFFromStream[210];
- LoadRTFFromStream, LoadRTFFromStreamW[213].

1.4.1.1.3.90 TRichView.LoadRTF

Appends the content of RTF (Rich Text Format) file **FileName** to the document.

```
function LoadRTF(const FileName: String):Boolean;
```

(Introduced in version 1.5)

Despite its name, this method does not clear the existing document before loading. To replace the current document with the content of this file, call Clear[175] method before loading (and may be DeleteUnusedStyles[181] after Clear)

Method type: viewer-style.

Parameters for loading are in RTFReadProperties[151]. The most important properties affecting RTF loading are RTFReadProperties.TextStyleMode[274] and ParaStyleMode[272]. The imported RTF may look absolutely different depending on values of these properties! Setting for RTF loading can be changed in the TRichView component editor[134].

See important note on loading external images from RTF[672].

By default, hyperlink targets are loaded in items tags[34]. You can customize loading using OnReadHyperlink[248] event.

If style templates are used[158], and RTFReadProperties.TextStyleMode=RTFReadProperties.ParaStyleMode=*rvrsAddIfNeeded*, and StyleTemplateInsertMode[156]<>*rvstimIgnoreSourceStyleTemplates*, the method loads an RTF style sheet into Style[156].StyleTemplates[402]. If TRichView was completely empty before loading, LoadRTF replaces the existing style templates. Otherwise, it merges the RTF style sheet into the style templates, and reads text and paragraph styles according to StyleTemplateInsertMode[156]. The method calls OnStyleTemplatesChange[266] event.

Return value: "Was reading successful?" (extended information is in RTFReadProperties.ErrorCode[270])

Unicode note: Text can be imported as Unicode or not, depending on value of RTFReadProperties.UnicodeMode[274].

See also methods of TRichView:

- LoadRTFFromStream[210];
- SaveRTF[219].

See also events of TRichView:

- OnImportPicture[242];
- OnReadHyperlink[248];
- OnStyleTemplatesChange[266];

- OnProgress 248.

See also methods of TRichViewEdit:

- InsertRTFFromFileEd 323.

See also:

- Saving and loading in RichView 58.

1.4.1.1.3.91 TRichView .LoadRTFFromStream

Appends the content of RTF (Rich Text Format) stream **Stream** to the document.

```
function LoadRTFFromStream(Stream: TStream):Boolean;
```

(Introduced in version 1.5)

Despite its name, this method does not clear the existing document before loading. To replace the current document with the content of this file, call Clear 175 method before loading (and may be DeleteUnusedStyles 181 after Clear)

Method type: viewer-style.

Parameters for loading are in TRichView.RTFReadProperties 151. The most important properties affecting RTF loading are RTFReadProperties.TextStyleMode 274 and ParaStyleMode 272. The imported RTF may look absolutely different depending on values of these properties! Setting for RTF loading can be changed in the TRichView component editor 134.

See important note on loading external images from RTF 672.

By default, hyperlink targets are loaded in items tags 34. You can customize loading using OnReadHyperlink 248 event.

If style templates are used 158, and RTFReadProperties.TextStyleMode=RTFReadProperties.ParaStyleMode=*rvrsAddIfNeeded*, and StyleTemplateInsertMode 156<>*rvstimIgnoreSourceStyleTemplates*, the method loads an RTF style sheet into Style 156.StyleTemplates 402. If TRichView was completely empty before loading, LoadRTFFromStream replaces the existing style templates. Otherwise, it merges the RTF style sheet into the style templates, and reads text and paragraph styles according to StyleTemplateInsertMode 156. The method calls OnStyleTemplatesChange 266 event.

Return value: "Was reading successful?" (extended information is in TRichView.RTFReadProperties.ErrorCode 270)

See also methods of TRichView:

- LoadRTF 209;
- SaveRTFToStream 220.

See also events of TRichView:

- OnImportPicture 242;
- OnReadHyperlink 248;
- OnStyleTemplatesChange 266;
- OnProgress 248.

See also methods of TRichViewEdit:

- InsertRTFFromStreamEd 323.

See also:

- Saving and loading in RichView 58.

1.4.1.1.3.92 TRichView.LoadRVF

Clears the document and loads a new document from the file **FileName** as RVF (RichView Format 59).

```
function LoadRVF(const FileName: String): Boolean;
```

Method type: viewer-style.

Setting for RVF loading can be changed in the TRichView component editor 134.

Return value: "Was reading successful?"

See also properties:

- RVFOptions 152;
- RVFWarnings 155;
- RVFTextStylesReadMode 154 (this method type: RVF loading);
- RVFParaStylesReadMode 154 (this method type: RVF loading);
- UseStyleTemplates 158.

See also events:

- OnRVFImageListNeeded 252;
- OnRVFControlNeeded 251;
- OnRVFPictureNeeded 252;
- OnControlAction 239 (ControlAction=*rvcaAfterRVFLoad*);
- OnStyleTemplatesChange 266;
- OnProgress 248.

See also methods:

- LoadRVFFromStream 211;
- AppendRVFFromStream 173;
- InsertRVFFromStream 206;
- Format 182;
- FormatTail 183.

See also methods of TRichViewEdit:

- InsertRVFFromFileEd 324.
- **See also:**
- TRichView component editor 134;
- Saving and loading 58;
- RVF overview 59.

1.4.1.1.3.93 TRichView.LoadRVFFromStream

Clears the document and loads a new document from the file stream **Stream** as RVF (RichView Format 59).

```
function LoadRVFFromStream(Stream: TStream): Boolean;
```

Method type: viewer-style.

Setting for RVF loading can be changed in the TRichView component editor 134.

Return value: "Was reading successful?"

See also properties:

- RVFOptions 152;
- RVFWarnings 155;
- RVFTextStylesReadMode 154 (this method type: RVF loading);
- RVFParaStylesReadMode 154 (this method type: RVF loading);
- UseStyleTemplates 158.

See also events:

- OnRVFImageListNeeded 252;

- OnRVFControlNeeded[251];
- OnRVFPictureNeeded[252];
- OnControlAction[239] (ControlAction=*rvcaAfterRVFLoad*);
- OnStyleTemplatesChange[266];
- OnProgress[248].

See also methods:

- LoadRVF[211];
- AppendRVFFromStream[173];
- InsertRVFFromStream[206];
- Format[182];
- FormatTail[183].

See also methods of TRichViewEdit:

- InsertRVFFromFileEd[324].
- **See also:**
- TRichView component editor[134];
- Saving and loading[58];
- RVF overview[59].

1.4.1.1.3.94 TRichView .LoadText -W

Append the content of the text file **FileName** to the document. **LoadText** loads text from ANSI text files, **LoadTextW** loads text from Unicode (UTF-16) text files.

```
function LoadText(const FileName: String; StyleNo, ParaNo: Integer;
  AsSingleParagraph: Boolean; CodePage: Cardinal=CP_ACP):Boolean;

function LoadTextW(const FileName: String; StyleNo, ParaNo: Integer;
  DefAsSingleParagraph: Boolean):Boolean;
```

Despite their names, these methods do not clear the existing document before loading. To replace the current document with the content of this file, call Clear[175] method before loading (and may be DeleteUnusedStyles[181] after Clear).

Parameters

StyleNo – an index of text style[404] which will be used for new text. It defines font attributes for the added text.

ParaNo – an index of paragraph style[398] which will be used for new text. It defines paragraph attributes for the added text.

AsSingleParagraph (and **DefAsSingleParagraph**):

- *False* – the methods use the current add-paragraph-mode (see SetAddParagraphMode[225]);
- *True* – the methods add all text as a single paragraph (add the first new item in the current add-paragraph-mode, set add-paragraph-mode to *False*, add the rest of text, restore the add-paragraph-mode). **DefAsSingleParagraph** only affects processing CR and LF characters. WideChar($2029) ("paragraph separator") always initiates a new paragraph. WideChar($2028) ("line separator") always initiates a new line.

CodePage – a code page for converting from ANSI to Unicode. It is used only if **StyleNo** points to a Unicode style; otherwise it is ignored.

LoadTextW supports the Unicode byte order marks characters (if they are present, they must be the first characters in the file).

Methods type: viewer-style.

Unicode notes:

LoadText (text in the file must be ANSI):

- If the **StyleNo**-th style is ANSI style, text from the file is loaded as it is.
- If the **StyleNo**-th style is Unicode style, text from the file is converted to Unicode (conversion is performed using the **CodePage** parameter, or, of it's equal to CP_ACP, basing on the Charset[434] of this style).

LoadTextW (text in the file must be Unicode):

- If the **StyleNo**-th style is Unicode style, text from the file is added as it is..
- If the **StyleNo**-th style is ANSI style, text from the file is converted to ANSI (conversion is performed basing on the Charset 434 of this style).

If style templates 158 are used, the component may automatically change text style indices of text items, to provide a consistency of text and paragraph style templates.

Return value:"Was loading successful?"

See also methods:

- LoadTextFromStream, LoadTextFromStreamW 213;
- AddTextNLA, AddTextNLW 171;
- Format 182;
- FormatTail 183;
- SetAddParagraphMode 225;
- SaveText, SaveTextW 221.

See also methods of TRichViewEdit:

- InsertTextFromFile, InsertTextFromFileW 328.

See also properties of TRVStyle:

- TextStyles 404;
- SpacesInTab 402.

See also:

- Other methods for appending items 43;
- Saving and loading 58;
- Unicode in RichView 64;
- Example how to load UTF-8 files 282.

1.4.1.1.3.95 TRichView.LoadTextFromStream-W

Append the content of the stream **Stream** to the document. **LoadTextFromStream** reads ANSI text, **LoadTextFromStreamW** reads Unicode (UTF-16) text.

```
function LoadTextFromStream(Stream: TStream; StyleNo, ParaNo: Integer;
  AsSingleParagraph: Boolean; CodePage: Cardinal=CP_ACP):Boolean;

function LoadTextFromStreamW(Stream: TStream; StyleNo, ParaNo: Integer;
  DefAsSingleParagraph: Boolean):Boolean;
```

The same as LoadText and LoadTextW 212, but read text from streams instead of files.

1.4.1.1.3.96 TRichView.MarkStylesInUse

Stores lists of used text 404, paragraph 398 and list 396 styles.

```
procedure MarkStylesInUse(Data: TRVDeleteUnusedStylesData 612);
```

(introduced in version 10)

This method is useful if collections of text, paragraph and list styles are not saved in RVF documents 59 (but stored in file or in the Registry). In this case, several documents share the same collections of styles. See TRVDeleteUnusedStylesData 612 for the examples.

Usually it's much more simple (and recommended) to use one RVStyle per one RichView and store collections of styles inside RVF documents. In this case, you can use DeleteUnusedStyles 181 instead.

See also methods:

- DeleteMarkedStyles 180;
- DeleteUnusedStyles 181.

1.4.1.1.3.97 TRichView.Reformat

Reformats the document.

Only formatted documents can be displayed or edited.

```
procedure Reformat;
```

(introduced in v1.9)

This method works like Format 182, but keeps the selection (and caret position in TRichViewEdit 282).

Document must be formatted before calling this method.

This method is useful, for example, if you want to reformat document after changing margins.

1.4.1.1.3.98 TRichView.RefreshListMarkers

Recalculates list markers 99

```
procedure RefreshListMarkers;
```

(introduced in v1.8)

Call it (before calling Format 182) if you changed ListType 476 or FormatString 474 of list style level of marker that is already inserted in TRichView. It's usually not necessary to use this method.

1.4.1.1.3.99 TRichView.RefreshListSequences

Recalculates numbered sequences 102, footnotes 104, endnotes 103, and sidenotes 105.

```
procedure RefreshSequences;
```

(introduced in v13)

Call it (before calling Format 182) if you changed FootnoteNumbering 389, EndnoteNumbering 389 or SidenoteNumbering 389 of TRVStyle 385 component linked 156 to this TRichView control, or a property of a numbered sequence item that was already inserted, to update existing document containing footnotes or endnotes.

1.4.1.1.3.100 TRichView.RemoveCheckpoint

Removes the *checkpoint* associated with the **ItemNo**-th item.

```
function RemoveCheckpoint(ItemNo: Integer): Boolean;
```

ItemNo – index of the item. Items are indexed from 0 to ItemCount 144-1. Items of subdocuments (table cells 567) are not included in the items range of the main document; for items in cells, use Cell.GetRVData 575.RemoveCheckpoint.

If the **ItemNo**-th item does not have a *checkpoint*, this method does nothing.

Methods type: viewer-style.

Repainting may be needed if *rvoShowCheckpoints* in Options 146.

See also methods of TRichView:

- GetItemCheckpoint 191;
- SetCheckpointInfo 227.

See also properties:

- ItemCount 144.

See also methods of RichViewEdit:

- RemoveCheckpointEd 333;
- RemoveCurrentCheckpoint 334.

See also:

- "Checkpoints 30";
- Modifying RichView items 51.

1.4.1.1.3.101 TRichView.ResetAnimation

Rewinds all inserted animated pictures to the first frame.

```
procedure ResetAnimation;
```

(introduced in version 10)

See also methods:

- StartAnimation 237;
- StopAnimation 237.

See also properties:

- AnimationMode 135.

See also:

- Animation in TRichView 56;
- Pictures 90;
- *Hot-Pictures* 93.

1.4.1.1.3.102 TRichView.SaveDocX

Exports TRichView document (or the selected part, if **SelectionOnly**=*True*) to the file **FileName** as DocX (Microsoft Word format).

```
function SaveDocX(const FileName: String; SelectionOnly: Boolean):Boolean;
```

(introduced in version 15)

This method is available for Delphi 4+, C++Builder 2006+

The method uses *rvrtfSaveDocParameters* and *rvrtfSaveHeaderFooter* options from RTFOptions 150 property.

By default, inserted controls are not saved. You can save them yourself using OnSaveComponentToFile 256 event.

By default, items tags 34 are saved as targets of hypertext links. You can customize saving using OnWriteHyperlink 267 event.

You can:

- completely change DocX code for certain items using OnSaveItemToFile 261 event;
- insert OOXML codes 438 in text;
- insert additional code in DocX in OnSaveDocXExtra 257 event.

Return value: "successful saving?"

See also methods:

- SaveDocXToStream 216;

See also properties:

- RTFOptions 150;
- UseStyleTemplates 158.

See also events:

- OnWriteHyperlink 267;

- OnSaveComponentToFile 256;
- OnSaveDocXExtra 257;
- OnSaveItemToFile 261;
- OnProgress 248.

See also global variables and constants:

- RichViewSaveHyperlinksInDocXAsFields 656

See also:

- Saving and loading in RichView 58.

1.4.1.1.3.103 TRichView.SaveDocXToStream

Exports TRichView document (or the selected part, if **SelectionOnly**=*True*) to the stream **Stream** as DocX (Microsoft Word format).

```
function SaveDocXToStream(Stream: TStream; SelectionOnly: Boolean):Boolean;
```

(introduced in version 15)

This method is available for Delphi 4+, C++Builder 2006+

The same as SaveDocX 215, but saves to a stream instead of a file.

1.4.1.1.3.104 TRichView.SaveHTML

Exports TRichView document to HTML or XHTML file, using HTML tags like <font>, <b>, , etc.

```
function SaveHTML(const FileName, Title, ImagesPrefix: String;
  Options: TRVSaveOptions 639): Boolean;
```

This method saves HTML file and a set of images (in separate files).

Files saved by this method can be rendered by the most of HTML applications. But many advanced formatting options are lost. Files saved by SaveHTMLEx 217 look better in modern browsers.

Parameters:

FileName – the name of the output HTML file.

Title – the title of the output HTML file.

ImagesPrefix – the first part of names of images that will be saved with HTML document.

Options – options for saving, see TRVSaveOptions 639 for possible values.

By default, inserted controls are not saved. You can save them yourself using OnSaveComponentToFile 256 event.

By default, items tags 34 are saved as targets of hypertext links. You can customize saving using OnWriteHyperlink 267 event.

You can:

- completely change HTML code for certain items using OnSaveItemToFile 261 event;
- insert additional code in HTML in OnSaveHTMLExtra 259 and OnSaveParaToHTML 263 events;
- change how images are saved using OnHTMLSaveImage 241 or OnSaveImage2 260 event,
- insert HTML codes 438 in text.

Return value: "successful saving?"

See also methods:

- SaveHTMLEx 217;
- SaveHTMLToStream 218.

See also events:

- OnWriteHyperlink 267;
- OnSaveComponentToFile 256;
- OnSaveHTMLExtra 259;
- OnHTMLSaveImage 241;
- OnSaveImage2 260;
- OnSaveItemToFile 261;
- OnSaveParaToHTML 263;
- OnProgress 248.

See also:

- Export to HTML 62;
- RichViewSavePlnHTML 652 typed constant;
- TFontInfo.Options 444 (HTML codes);
- Saving and loading 58.

1.4.1.1.3.105 TRichView.SaveHTMLEx

Exports TRichView document to HTML or XHTML file, using CSS (Cascading Style Sheets)

```
function SaveHTMLEx(const FileName, Title, ImagesPrefix, ExtraStyles,
  ExternalCSS, CPPrefix: String;
  Options: TRVSaveOptions 639): Boolean;
```

This method saves HTML file and a set of images (in separate files).

Parameters:

FileName –- the name of the output HTML file.

Title –- the title of the output HTML file.

ImagesPrefix –- the first part of names of images that will be saved with HTML document;

ExtraStyles –- strings that can contain additional entries of CSS (usually you need not to use it, set to '').

ExternalCSS –- if this string is not empty, this method uses external CSS (saved with RichView.Style.SaveCSS 408) instead of saving CSS into HTML file).

CPPrefix –- first part of *checkpoints* 30 names when saving to HTML; set to '' for using default names ('RichViewCheckPoint'). This parameter is not used, if *rvsoUseCheckpointsNames* is in Options.

Options – options for saving, see TRVSaveOptions 639 for possible values.

By default, inserted controls are not saved. You can save them yourself using OnSaveComponentToFile 256 event.

By default, items tags 34 are saved as targets of hypertext links. You can customize saving using OnWriteHyperlink 267 event.

You can:

- completely change HTML code for certain items using OnSaveItemToFile 261 event;
- insert additional code in HTML in OnSaveHTMLExtra 259 and OnSaveParaToHTML 263 events;
- change how images are saved using OnHTMLSaveImage 241 or OnSaveImage2 260 event,
- insert HTML codes 438 in text.

Since v1.9, the resulting HTML conforms "HTML 4.01 Transitional" standard, see http://validator.w3.org/.

Two issues may cause the validator's warnings:

- HTML encoding is written in HTML only if Charset 434 of the 0-th text style <> DEFAULT_CHARSET;
- non-standard <TABLE bordercolor> attribute is saved for MS Internet Explorer.

Return value: "successful saving?"

See also methods:

- SaveHTML 216;
- SaveHTMLToStreamEx 218.

See also events:

- OnWriteHyperlink 267;
- OnSaveComponentToFile 256;
- OnSaveHTMLExtra 259;
- OnHTMLSaveImage 241;
- OnSaveImage2 260;
- OnSaveItemToFile 261;
- OnSaveParaToHTML 263;
- OnProgress 248.

See also:

- Export to HTML 62;
- TFontInfo.Options 444 (HTML codes);
- RichViewSaveDivInHTMLEx 652 typed constant;
- Saving and loading 58.

1.4.1.1.3.106 TRichView.SaveHTMLToStream

Exports TRichView document as HTML or XHTML to the stream **Stream**, using HTML tags like <font>, <b>, , etc.

```
function SaveHTMLToStream(Stream: TStream; const Path, Title,ImagesPrefix: String;
  Options: TRVSaveOptions 639): Boolean;
```

This method saves HTML to **Stream** and creates files for images.

The same as SaveHTML 216, but saves to a stream instead of a file.

Parameters:

Stream –- stream where to export HTML.

Path –- path where to save images (must have '\' as the last character).

For the descriptions of other parameters, see SaveHTML 216.

1.4.1.1.3.107 TRichView.SaveHTMLToStreamEx

Exports contents of RichView to stream as HTML or XHTML, using CSS (Cascading Style Sheets)

```
function SaveHTMLToStreamEx(Stream: TStream;
  const Path, Title, ImagesPrefix, ExtraStyles,
  ExternalCSS, CPPrefix: String;
  Options: TRVSaveOptions 639): Boolean;
```

This method saves HTML to **Stream** and creates files for images.

The same as SaveHTMLEx 217, but saves to a stream instead of a file.

Parameters:

Stream –- stream where to export HTML.

Path –- path where to save images (must have '\' as the last character).

For the descriptions of other parameters, see SaveHTMLEx 217.

1.4.1.1.3.108 TRichView.SavePicture

Saves picture. This method is used when saving HTML files.

```
function SavePicture(DocumentSaveFormat: TRVSaveFormat[639];
  const Path: String; gr: TGraphic): String; virtual;
```

Parameters:

gr – image to save.

DocumentSaveFormat – output document format, see TRVSaveFormat[639] for possible values.

Path – path where to save the image (the last character must be '\').

If **DocumentSaveFormat**=*rvsfHTML* then the image is saved as Jpeg image (or as bitmap, if the application was compiled with RVDONOTUSEJPEGIMAGE conditional define). You can define image formats that will not be converted to Jpeg[672].

If **DocumentSaveFormat**<>*rvsfHTML*, the image is saved as bitmap.

This method must be used only inside the following events:

- OnHTMLSaveImage[241],
- OnSaveImage2[260],
- OnSaveComponentToFile[256],
- OnSaveItemToFile[261] (SaveFormat=rvsfHTML).

The image file name will consist of the image prefix (see the **ImagesPrefix** parameter of SaveHTML[216] and SaveHTMLEx[217]) and number. The file name may be unique or not (depending on *rvsoOverrideImages* in the **Options** parameter of SaveHTML[216] and SaveHTMLEx[217]).

See also:

- Export to HTML[62].

1.4.1.1.3.109 TRichView.SaveRTF

Exports TRichView document (or the selected part, if **SelectionOnly**=*True*) to the file **FileName** as RTF (Rich Text Format).

```
function SaveRTF(const FileName: String; SelectionOnly: Boolean):Boolean;
```

(introduced in version 1.3)

Options for saving are in RTFOptions[150] property.

By default, inserted controls are not saved. You can save them yourself using OnSaveComponentToFile[256] event.

By default, items tags[34] are saved as targets of hypertext links. You can customize saving using OnWriteHyperlink[267] event.

You can:

- completely change RTF code for certain items using OnSaveItemToFile[261] event;
- insert additional code in RTF in OnSaveRTFExtra[263] event;
- insert RTF codes[438] in text.

Return value: "successful saving?"

See also methods:

- SaveRTFToStream[220];

- CopyRTF 178;
- LoadRTF 209.

See also properties:

- RTFOptions 150;
- UseStyleTemplates 158.

See also events:

- OnWriteHyperlink 267;
- OnSaveComponentToFile 256;
- OnSaveRTFExtra 263;
- OnSaveItemToFile 261;
- OnProgress 248.

See also:

- Saving and loading in RichView 58.

1.4.1.1.3.110 TRichView.SaveRTFToStream

Exports TRichView document (or the selected part, if **SelectionOnly**=*True*) to the **Stream** as RTF (Rich Text Format)

```
function SaveRTFToStream(Stream: TStream; SelectionOnly: Boolean):Boolean;
```

The same as SaveRTF 219, but saves to a stream instead of a file.

1.4.1.1.3.111 TRichView.SaveRVF

Saves the document (or the selected part, if **SelectionOnly**=*True*) to the file **FileName** as RVF (RichView Format 59)

```
function SaveRVF(const FileName: String; SelectionOnly: Boolean): Boolean;
```

Setting for RVF saving can be changed in the TRichView component editor 134.

Return value: "successful saving?"

See also properties:

- RVFOptions 152.

See also methods:

- SaveRVFToStream 220;
- LoadRVF 211.

See also:

- Saving and loading RichView document 58;
- RVF 59.

1.4.1.1.3.112 TRichView.SaveRVFToStream

Saves the document (or the selected part, if **SelectionOnly**=*True*) to the **Stream** as RVF (RichView Format 59).

```
function SaveRVFToStream(Stream: TStream; SelectionOnly: Boolean): Boolean;
```

Setting for RVF saving can be changed in the TRichView component editor 134.

Return value: "successful saving?"

See also properties:

- RVFOptions 152.

See also methods:

- SaveRVF 220;
- LoadRVFFromStream 211;
- InsertRVFFromStream 206;
- AppendRVFFromStream 173.

See also:

- Saving and loading RichView document 58;
- RVF 59.

1.4.1.1.3.113 TRichView.SaveText -W

The methods export document to ANSI (**SaveText**) or Unicode (**SaveTextW**) text file.

```
function SaveText(const FileName: String; LineWidth: Integer; CodePage: Cardinal=CP_ACP):
function SaveTextW(const FileName: String; LineWidth: Integer): Boolean;
```

Parameters:

FileName – output file name.

LineWidth is used for saving *breaks* 93 (they are saved as **LineWidth** '-' characters)

Unicode notes:.

SaveText: text of Unicode styles is converted using the **CodePage** parameter (or, if **CodePage**=CP_ACP, using TRVStyle.DefCodePage 388).

SaveTextW: text of ANSI styles is converted basing on its character sets 434 (if Charset=DEFAULT_CHARSET, it is converted using TRVStyle.DefCodePage 388).

Return value: "successful saving?"

See also methods:

- SaveTextToStream, SaveTextToStreamW 221;
- LoadText, LoadTextW 212.

See also events:

- OnSaveImage2 260;
- OnSaveComponentToFile 256;
- OnSaveItemToFile 261.

See also:

- Saving and loading RichView documents 58.

1.4.1.1.3.114 TRichView.SaveTextToStream -W

The methods export document or selection as ANSI (**SaveTextToStream**) or Unicode (**SaveTextToStreamW**) text.

```
function SaveTextToStream(const Path: String; Stream: TStream;
  LineWidth: Integer; SelectionOnly, TextOnly: Boolean;
  CodePage: Cardinal=CP_ACP):Boolean;
function SaveTextToStreamW(const Path: String; Stream: TStream;
  LineWidth: Integer; SelectionOnly, TextOnly: Boolean):Boolean;
```

(introduced in version 1.3)

Parameters

LineWidth is used for saving *breaks* 93 (they are saved as **LineWidth** '-' characters)

Path – path for saving images and other non-text items (they are not saved by default, but this parameter is

used in OnSaveImage2 260, OnSaveComponentToFile 256, and OnSaveItemToFile 261 events);

SelectionOnly – if *True*, only selected part of the document is saved.

TextOnly – if *True*, non-text items are ignored when saving. If *False*, text representation of items is saved.

Unicode notes:.

SaveTextToStream: text of Unicode styles is converted using the **CodePage** parameter (or, if **CodePage**=CP_ACP, using TRVStyle.DefCodePage 388).

SaveTextToStreamW: text of ANSI styles is converted basing on its character sets 434 (if Charset=DEFAULT_CHARSET, it is converted using TRVStyle.DefCodePage 388).

Return value: "successful saving?"

See also methods:

- SaveText, SaveTextW 221;
- LoadTextFromStream, LoadTextFromStreamW 213.

See also events:

- OnSaveImage2 260;
- OnSaveComponentToFile 256;
- OnSaveItemToFile 261.

See also:

- Saving and loading RichView documents 58.

1.4.1.1.3.115 TRichView .SearchText -A -W

The methods search for the substring **s** in the document.

```
function SearchText(s: String; SrchOptions: TRVSearchOptions 642): Boolean;
function SearchTextA(s: TRVAnsiString 623; SrchOptions: TRVSearchOptions 642): Boolean;
function SearchTextW(s: TRVUnicodeString 647; SrchOptions: TRVSearchOptions 642): Boolean;
```

When found, these methods select substring and scroll to it.

if *rvsroDown* is not included in **SrchOptions**, the methods search upwards. if *rvsroDown* is included, the methods search downwards.

If *rvsroFromStart* is included in **SrchOptions**, the search is started from the beginning/end of the document. Otherwise, the methods start searching from the current selection, if it exists. If nothing is selected, the methods start from the first visible item (when searching down) or the last visible item (when searching up).

rvsroSmartStart is used only if *rvsroFromStart* is not included. If *rvsroSmartStart* is included, and some document fragment is selected, then:

- when searching down, the search starts from the second selected character;
- when searching up, the search starts from the last but one selected character;

(selected characters are counted from the top of document, the selection direction is ignored). This option allows to avoid selecting the same fragment again when changing a search direction.

If *rvsroMultiItem* is included, the search can match substrings of several text items 88. If not included, the text is searched in each text item separately. For example, if the document contains the text **He*llo***, the substring 'Hello' can be found only if this option is included, because 'He' and 'llo' belong to different items.

Unicode note: The methods allow searching both in ANSI and in Unicode 64 text items. When **SearchTextA** compares Unicode text with **s**, the conversion is based on RVStyle.DefCodePage 388. When **SearchTextW** compares ANSI text with **s**, the conversion is based on the character set of the text style.

SearchText works like **SearchTextW** in Delphi/C++Builder 2009 or newer, and like **SearchTextA** in the older versions.

History and compatibility

Since version 1.2 SearchText can search upwards.
Since version 1.5 SearchText can search whole words (substring must be delimited with characters from Delimiters 140)
SearchTextW is introduced in version 10. rvsroFromStart is introduced in version 10 as well.
rvsroMultiItem is introduced in version 12.

Note: There is incompatibility with programs written with the older version of RichView (1.0 – 1.1.4). Now if rvsroDown is not included in SrchOptions, the methods search upwards. In the older versions there was no rvsroDown option, and SearchText searched downwards only.

These methods must be called only when the document is formatted.

Return value: Is substring found?

See also methods of TRichViewEdit:

- SearchText, -A, -W 336

See also global functions:

- GetRVSearchOptions 616.

See also:

- Searching in RichView and RichViewEdit 49.

1.4.1.1.3.116 TRichView.SelectAll

Selects the whole document for copying to the Clipboard

```
procedure SelectAll;
```

This method does not repaint document, call Invalidate.

In RichViewEdit 282 this method moves the caret to the end of document. The method generates OnSelect 265 event.

This method must be called only when the document is formatted.

See also methods:

- SetSelectionBounds 235;
- Deselect 181.

See also:

- Selecting in RichView document 47.

See also:

- Edit 610 method of table cell.

1.4.1.1.3.117 TRichView.SelectControl

Selects the given visual control in the document.

```
function SelectControl(actrl: TControl): Boolean;
```

(introduced in version 1.8)

The method selects the item 94 containing the control **actrl**.

This method repaints document.

The method generates OnSelect 265 event.

In TRichViewEdit 282, if this control's item is resizable (see *rvepResizable* 627 extra-property 627) and resizing with mouse is not forbidden (see *rvoNoImageResize* in EditorOptions 291), resizing handles appear. A good idea to call this method in OnClick event of resizable controls.

This method must be called only when the document is formatted.

Return value: *True* if this control was found and selected.

See also methods:

- SetSelectionBounds 235;
- Deselect 181.

See also:

Selecting in RichView document 47.

1.4.1.1.3.118 TRichView.SelectionExists

"Is some part of document selected for copying to the Clipboard?"

```
function SelectionExists: Boolean;
```

Note: In some cases, this function can generate OnSelect 265 event, but it's safe to use it inside OnSelect event handler (infinite recursion will not occur)

This method must be called only when the document is formatted.

See also:

- Selecting in RichView document 47.

1.4.1.1.3.119 TRichView.SelectWordAt

Selects word at the specified point for copying to clipboard. If there is non text item at this point (such as picture), this item becomes selected.

```
procedure SelectWordAt(X,Y: Integer);
```

"Word" is defined as a part of string between delimiters. Delimiters are listed in Delimiters 140 property.

This method repaints document.

In RichViewEdit 282 this method moves the caret to the end of the selected word.

The method generates OnSelect 265 event.

Parameters:

(**X**,**Y**) – client coordinates (coordinates relative to the top left corner of RichView window).

This method must be called only when the document is formatted.

Note: RichView can automatically select double-clicked word, if *rvoDblClickSelectsWord* is in Options 146.

See also events:

- OnRVRightClick[255];
- OnRVDblClick[250].

See also properties:

- Delimiters[140].

See also methods:

- SelectAll[223];
- GetWordAt[205];
- SetSelectionBounds[235];
- GetSelText[204].

See also methods of TRichViewEdit:

- SelectCurrentWord[337].

See also:

- Selecting in RichView document[47].

1.4.1.1.3.120 TRichView.SetAddParagraphMode

Sets mode for adding items from new line for all Add*** methods[43].

```
procedure SetAddParagraphMode(AllowNewPara: Boolean);
```

(Introduced in version 1.2)

By default (if **SetAddParagraphMode** was not called), all items that will be added from new line will start new paragraphs.

You can change this mode calling **SetAddParagraphMode**(False), and then return this mode again calling **SetAddParagraphMode**(True).

After **SetAddParagraphMode**(False) items that will be added from new line will not start new paragraphs but will be displayed from new line inside the previous paragraph.

Warning: be careful using **SetAddParagraphMode**(False):

- do not add the first item (or the item after *breaks*[93] or tables[98]) in this mode;
- do not switch paragraph style in this mode (all items in the same paragraph must have the same paragraph style).

See: valid documents[74]

Example 1:

```
with MyRichView do begin
  Clear[175];
  SetAddParagraphMode(True); // default mode
  AddNL[167]('First',  0, 0); // ok, starting 1st paragraph with style 0
  AddNL('Second', 0, 0); // ok, starting 2nd paragraph with style 0
  AddNL('Third',  0, 1); // ok, starting 3rd paragraph with style 1
end;
```

Example 2:

```
with MyRichView do begin
  Clear;
  SetAddParagraphMode(False);
  AddNL('First', 0, 0); // error, do not add the first item in this mode!
end;
```

Example 3:

```
with MyRichView do begin
  Clear;
```

```
  SetAddParagraphMode(True); // default mode
  AddNL('First',  0, 0); // ok, starting 1st paragraph with style 0
  SetAddParagraphMode(False);
  AddNL('Second', 0, 0); // ok, adding this item from new line inside the 1st paragraph
  AddNL('Third',  0, 1); // error, do not add items with another paragraph style in this mo
end;
```

See also

- Paragraphs 37;
- Building a document 43.

1.4.1.1.3.121 TRichView .SetBreakInfo

Changes main properties of the item of *break* 93 (horizontal line) type.

```
procedure SetBreakInfo(ItemNo: Integer; AWidth: TRVStyleLength644;
  AStyle: TRVBreakStyle624; AColor: TColor; const ATag: TRVTag645);
```

Parameters:

ItemNo – index of the item. The item must be of *break* 93 type (*rvsBreak* 663), otherwise the method raises ERichViewError 608 exception. Items are indexed from 0 to ItemCount 144-1, GetItemStyle 194 returns type of item. Items of subdocuments (table cells 567) are not included in the items range of the main document; for items in cells, use Cell.GetRVData 575.SetBreakInfo.

AWidth – line width (or rectangle height). This value is measured in Style 156.Units 404.

AStyle – visual style of this *break*, see TRVBreakStyle 624 for possible values.

AColor – line color. If it is equal to *clNone*, Style 156.TextStyles 404[0].Color 435 is used.

ATag – *tag* of the item.You can use value returned by GetBreakInfo 183 or GetItemTag 195 for this item.

Instead of this method, you can use SetItemExtraIntPropertyEx 230 and SetItemTag 231 methods.

Method type: viewer-style. It's not necessary to reformat document after it, repainting is enough.

Additional item properties are assigned by the methods SetItemExtraIntProperty 230 and SetItemExtraStrProperty 231.

Example

```
RichView2.SetBreakInfo(ItemNo, 1, rvbsLine, clNone, 'Break Tag');
```

See also methods:

- GetBreakInfo 183;
- GetItemStyle 194;
- SetItemExtraIntProperty 230;
- SetItemExtraStrProperty 231.

See also properties:

- ItemCount 144.

See also methods of RichViewEdit:

- SetBreakInfoEd 339;
- SetCurrentBreakInfo 342.

See also:

- Modifying RichView items 51;
- Item types 29;
- *Tags* 34.

1.4.1.1.3.122 TRichView.SetBulletInfo

Changes main properties of the item of *bullet* 96 (image from ImageList) type.

```
procedure SetBulletInfo(ItemNo: Integer; const AName: TRVAnsiString 623;
  AImageIndex: Integer; AImageList: TCustomImageList;
  const ATag: TRVTag 645);
```

Parameters:

ItemNo – index of the item. The item must be of *bullet* 96 or *hotspot* 97 type (*rvsBullet or rvsHotspot* 663), otherwise the method raises ERichViewError 608 exception. Items are indexed from 0 to ItemCount 144-1, GetItemStyle 194 returns type of item. Items of subdocuments (table cells 567) are not included in the items range of the main document; for items in cells, use Cell.GetRVData 575.SetBulletInfo.

AName – name of *bullet* 96, any string without line break (CR, LF) characters. It can also be set using SetItemText 232 method.

AImageList – not used, reserved, set it to *nil*.

AImageIndex – index of image in image list. It can also be set using SetItemExtraIntPropertyEx 230 method.

ATag – *tag* of the item. You can use value returned by GetBulletInfo 184 or GetItemTag 195 for this item. The *tag* can also be set by SetItemTag 231 method.

Method type: viewer-style. It's not necessary to reformat document after it, repainting is enough.

Additional item properties are assigned by the methods SetItemExtraIntProperty 230 and SetItemExtraStrProperty 231.

Unicode note: names of non-text items are ANSI strings.

Example

```
RichView1.SetBulletInfo(ItemNo, 'my image', 0, nil, 'bullet tag');
```

See also methods:

- GetBulletInfo 184;
- GetItemStyle 194;
- SetItemExtraIntProperty 230;
- SetItemExtraStrProperty 231.

See also properties:

- ItemCount 144.

See also methods of RichViewEdit:

- SetBulletInfoEd 339;
- SetCurrentBulletInfo 342.

See also:

- Modifying RichView items 51;
- Item types 29;
- *Tags* 34.

1.4.1.1.3.123 TRichView.SetCheckpointInfo

Creates a new *checkpoint* or modifies the existing *checkpoint* associated with the **ItemNo**-th item

```
procedure SetCheckpointInfo(ItemNo: Integer; const ATag: TRVTag 645;
  const AName: String; ARaiseEvent: Boolean);
```

Parameters

ItemNo – index of the item, from 0 to ItemCount 144-1. Items of subdocuments (table cells 567) are not included

in the items range of the main document; for items in cells, use Cell.GetRVData[575].SetCheckpointInfo. If this item already has a *checkpoint*, its properties are modified; otherwise, a new *checkpoint* is created.

ATag – *tag* of *checkpoint*;

AName – name of the *checkpoint*, any string without line break (CR, LF) characters.

ARaiseEvent – "raise event" flag; if set, RichView can generate OnCheckpointVisible[238] event for this *checkpoint*.

Method type: viewer-style. It's not necessary to reformat document after it, repainting is enough (repainting is needed if *rvoShowCheckpoints* in Options[146]).

Unicode note: names of *checkpoints* are Unicode strings for Delphi/C++Builder 2009 or newer, and ANSI strings for older versions of Delphi/C++Builder.

See also methods:

- GetItemCheckpoint[191];
- RemoveCheckpoint[214].

See also properties:

- ItemCount[144].

See also methods of RichViewEdit:

- SetCheckpointInfoEd[340];
- SetCurrentCheckpointInfo[343].

See also:

- Modifying RichView items[51];
- *Checkpoints*[30];
- *Tags*[34].

1.4.1.1.3.124 TRichView.SetControlInfo

Changes main properties of the item of control[94] type.

```
function SetControlInfo(ItemNo: Integer; const AName: TRVAnsiString[623];
  AVAlign: TRVVAlign[648]; const ATag: TRVTag[645]): Boolean;
```

Parameters:

ItemNo – index of the item. The item must be of control[94] type (*rvsComponent*[663]), otherwise the method raises ERichViewError[608] exception. Items are indexed from 0 to ItemCount[144]-1, GetItemStyle[194] returns type of item. Items of subdocuments (table cells[567]) are not included in the items range of the main document; for items in cells, use Cell.GetRVData[575].SetControlInfo.

AName – name of the item, any string without line break (CR, LF) characters. It can also be set using SetItemText[232] method. This is not a Name property of the control!

AVAlign – vertical alignment of this item, see TRVVAlign[648] for possible options.

ATag – *tag* of the item. Do not confuse with Tag property of the control! You can use value returned by GetControlInfo[187] or GetItemTag[195] for this item. The *tag* can also be set by SetItemTag[231] method.

Return value: "is reformatting needed? (i.e., is the new vertical align not equal to the old one?)"

Method type: viewer-style.

Additional item properties are assigned by the methods SetItemExtraIntProperty[230] and SetItemExtraStrProperty[231].

Unicode note: names of non-text items are ANSI strings.

Note: As you can see, current version of TRichView does not allow to change the control itself.

Example

```
RichView1.SetControlInfo(ItemNo, 'my control', rvvaBaseline, 'control tag');
```

See also methods:

- GetControlInfo 187;
- GetItemStyle 194;
- Format 182;
- SetItemExtraIntProperty 230;
- SetItemExtraStrProperty 231.

See also properties:

- ItemCount 144.

See also methods of RichViewEdit:

- SetControlInfoEd 341;
- SetCurrentControlInfo 344.

See also:

- Modifying RichView items 51;
- Item types 29;
- *Tags* 34.

1.4.1.1.3.125 TRichView.SetHotspotInfo

Changes main properties of the item of *hotspot* 97 (image from ImageList, hyperlink) type.

```
procedure SetHotspotInfo(ItemNo: Integer; const AName: TRVAnsiString 623;
  AImageIndex, AHotImageIndex: Integer;
  AImageList: TCustomImageList;
  const ATag: TRVTag 645);
```

Parameters:

ItemNo – index of the item. The item must be of *hotspot* 97 type (*rvsHotspot* 663), otherwise the method raises ERichViewError 608 exception. Items are indexed from 0 to ItemCount 144-1, GetItemStyle 194 returns type of item. Items of subdocuments (table cells 567) are not included in the items range of the main document; for items in cells, use Cell.GetRVData 575.SetHotspotInfo.

AName – name of *hotspot* 97, any string without line break (CR, LF) characters. It can also be set using SetItemText 232 method.

AImageList – not used, reserved, set it to *nil*..

AImageIndex – index of image in image list. It can also be set using SetItemExtraIntPropertyEx 230 method.

AImageIndex – index of "hot" image in image list. This image is displayed under the mouse pointer (in TRichView, or in TRichViewEdit in hypertext mode), or when user moves the caret to this item (in TRichViewEdit). It can also be set using SetItemExtraIntPropertyEx 230 method.

ATag – *tag* of the item. You can use value returned by GetHotspotInfo 189 or GetItemTag 195 for this item. The *tag* can also be set by SetItemTag 231 method.

Method type: viewer-style. It's not necessary to reformat document after it, repainting is enough.

Additional item properties are assigned by the methods SetItemExtraIntProperty 230 and SetItemExtraStrProperty 231.

Unicode note: names of non-text items are ANSI strings.

Example

```
RichView1.SetHotspotInfo(ItemNo, 'my image', 0, 0, nil, 'hotspot tag');
```

See also methods:

- GetHotspotInfo 189;
- GetItemStyle 194;
- SetItemExtraIntProperty 230;
- SetItemExtraStrProperty 231.

See also properties:

- ItemCount 144.

See also methods of RichViewEdit:

- SetHotspotInfoEd 349;
- SetCurrentHotspotInfo 345.

See also:

- Modifying RichView items 51;
- Item types 29;
- *Tags* 34.

1.4.1.1.3.126 TRichView .SetItemExtraIntProperty -Ex

The methods change a value of the specified integer property of the **ItemNo**-th item.

```
function SetItemExtraIntProperty(ItemNo: Integer;
  Prop: TRVExtraItemProperty 627; Value: Integer): Boolean;
function SetItemExtraIntProperty(ItemNo, Prop,
  Value: Integer): Boolean;
```

(introduced in versions 1.7 and 15)

These methods set a new **Value** of the item property identified by **Prop**.

ItemNo – index of the item, from 0 to ItemCount 144-1. Items of subdocuments (table cells 567) are not included in the items range of the main document; for items in cells, use Cell.GetRVData 575.SetItemExtraIntProperty[Ex]

In **SetItemExtraIntProperty**, **Prop**'s type is TRVExtraItemProperty 627. See information about this type for the list of properties.

In **SetItemExtraIntPropertyEx**, **Prop**'s type is Integer. If **Prop** can be converted to TRVExtraItemProperty, **SetItemExtraIntPropertyEx** works like **SetItemExtraIntProperty**. In addition, it supports properties identified by rveipc*** constants 658

Methods type: viewer-style. Reformatting or repainting may be needed, depending on the changed property.

Return value: *True*, if this item has this property. *False*, if not.

See also

- GetItemExtraIntProperty[Ex] 193;
- SetItemExtra 231**Str** 231Property 231;
- Format 182.

See also methods of TRichViewEdit

- SetItemExtraIntProperty[Ex]Ed 350;
- SetCurrentItemExtraIntProperty[Ex] 345.

1.4.1.1.3.127 TRichView .SetItemExtraStrProperty -Ex

The methods change a value of the specified additional string property of the **ItemNo**-th item.

```
function SetItemExtraStrProperty(ItemNo: Integer;
  Prop: TRVExtraItemStrProperty; const Value: String): Boolean;
function SetItemExtraStrProperty(ItemNo, Prop: Integer;
  const Value: String): Boolean;
```

(introduced in versions 1.9 and 15)

This methods set a new **Value** of the item property identified by **Prop**.

In **SetItemExtraStrProperty**, **Prop**'s type is TRVExtraItemStrProperty. See information about this type for the list of properties.

In **SetItemExtraStrPropertyEx**, **Prop**'s type is Integer. If **Prop** can be converted to TRVExtraItemStrProperty, **SetItemExtraStrPropertyEx** works like **SetItemExtraStrProperty**. In addition, it supports properties identified by rvespc*** constants

ItemNo – index of the item, from 0 to ItemCount-1. Items of subdocuments (table cells) are not included in the items range of the main document; for items in cells, use Cell.GetRVData .SetItemExtraStrProperty[Ex].

Method type: viewer-style. Reformatting or repainting may be needed, depending on the changed property.

Unicode note: string properties are Unicode strings for Delphi/C++Builder 2009 or newer, and ANSI strings for older versions of Delphi/C++Builder.

Return value: *True*, if this item has this property. *False*, if not.

See also:

- GetItemExtraStrProperty[Ex];
- SetItemExtra**Int**Property[Ex].

See also methods of TRichViewEdit:

- SetItemExtraStrProperty[Ex]Ed;
- SetCurrentItemExtraStrProperty[Ex].

1.4.1.1.3.128 TRichView .SetItemTag

Changing *tag* of the **ItemNo**-th item

```
procedure SetItemTag(ItemNo: Integer; const ATag: TRVTag);
```

Items are indexed from 0 to ItemCount-1. Items of subdocuments (table cells) are not included in the items range of the main document; for items in cells, use Cell.GetRVData.SetItemTag.

Method type: viewer-style. Reformatting or repainting is not necessary.

Example

```
RichView1.SetItemTag(ItemNo, 'tag #1');
```

See also methods:

- GetItemTag.

See also properties:

- ItemCount.

See also methods of RichViewEdit:

- SetItemTagEd 352;
- SetCurrentTag 349.

See also:

- Modifying RichView items 51;
- *Tags* 34.

1.4.1.1.3.129 TRichView .SetItemText -A -W

Assign text of text item or name of non-text item.

```
procedure SetItemText(ItemNo: Integer; const s: String);
procedure SetItemTextA(ItemNo: Integer; const s: TRVAnsiString 623);
procedure SetItemTextW(ItemNo: Integer; const s: TRVUnicodeString 647);
```

(Introduced in version 1.3, 1.7)

Parameters:

ItemNo – index of the item. Items are indexed from 0 to ItemCount 144-1, GetItemStyle 194 returns type of item. Items of subdocuments (table cells 567) are not included in the items range of the main document; for items in cells, use Cell.GetRVData 575.SetItemText*.

s – text assigned to the item. For text items, this is a visible text. For non-text item, this is a string value associated with the item (not displayed). **S** must not contain line break (CR and LF) characters. For text items, it must not contain CR, LF, TAB, FF characters (#13, #10, #9, #12).

Unicode notes:

SetItemTextA sets text from ANSI string:

- If the item is a non-text item or a text item of ANSI style, the text is assigned as it is.
- If the item is a Unicode text item, the function converts **s** to Unicode (conversion is based on Charset 434 of the item style).

SetItemTextW sets text from Unicode string:

- If the item is a Unicode text item, the text is assigned as it is.
- If the item is a non-text item or a text item of ANSI style, the function converts **s** to ANSI (conversion is based on Charset 434 of the item style, or on RVStyle.DefCodePage 388 for non-text items).

SetItemText works like **SetItemTextW** in Delphi/C++Builder 2009 or newer, and like **SetItemTextA** in the older versions of Delphi/C++Builder.

(There is one more version of this method, SetItemTextR, working with TRVRawByteString 638. In this method, the text is assigned as it is, without conversion. Be careful, incorrect text assignment may damage document).

Methods type: viewer-style. Reformatting and repainting are required only for text items.

See also methods:

- GetItemText -W -A 195;
- Format 182.

See also methods of TCustomRichViewEdit:

- SetItemTextEd -A -W 352;
- SetCurrentItemText -A -W 347.

See also:

- Modifying RichView Items 51;
- Unicode in RichView 64.

1.4.1.1.3.130 TRichView.SetItemVAlign

Changing vertical alignment (position relative to the line) of the **ItemNo**-th item

```
procedure SetItemVAlign(ItemNo: Integer; VAlign: TRVVAlign[648]);
```

(introduced in version 12)

Parameters:

ItemNo – index of the item. Items are indexed from 0 to ItemCount[144]-1. Items of subdocuments (table cells[567]) are not included in the items range of the main document; for items in cells, use Cell.GetRVData[575].SetItemVAlign.

VAlign – new value of vertical alignment.

This method must not be called for text items[88], tables[98], *breaks*[93], tabs[89], list markers[99], footnotes[104] and endnotes[103].

Method type: viewer-style. Requires reformatting and repainting.

See also methods:

- GetItemVAlign[196].

See also properties:

- ItemCount[144].

See also methods of RichViewEdit:

- SetItemVAlignEd[353];
- SetCurrentItemVAlign[348].

See also:

- Modifying RichView items[51]

1.4.1.1.3.131 TRichView.SetListMarkerInfo

Makes the given paragraph bulleted or numbered.

```
function SetListMarkerInfo(AItemNo, AListNo,
  AListLevel, AStartFrom, AParaNo: Integer;
  AUseStartFrom: Boolean): Integer;
```

(introduced in version 1.7)

Parameters:

AItemNo – index of the item, in range from 0 to ItemCount[144]-1. This is not necessary the index of list marker, it can be index of any item in the given paragraph. Items of subdocuments (table cells[567]) are not included in the items range of the main document; for items in cells, use Cell.GetRVData[575].SetListMarkerInfo. If this paragraph does not have list marker, a new list marker is created; otherwise, properties of the existing marker are modified.

If **AItemNo**<0 or >=ItemCount[144], a new marker is added at the end of the document. It's assumed that at least one item will be added after such marker in the same paragraph (see Valid Documents[74]).

AListNo – style of this list maker. This is an index in the collection of list styles (Style[156].ListStyles[396]).

AListLevel – list level. This is an index in the collection of list levels (Style.ListStyles[**AListNo**].Levels[459]).

AStartFrom: if **AUseStartFrom**=*True*, and this is a numbered list, this parameter is a value of list counter for this marker. If **AUseStartFrom**=*False*, numbering is continued.

AParaNo is used to define paragraph style of marker (if it is added as the last item, i.e. **AItemNo**<0). This is an index in the collection of paragraph styles (Style[156].ParaStyles[398]). If this marker is added to the existing paragraph, this parameter is ignored.

Method type: viewer-style.

Important note: list marker cannot be the only item in paragraph. It must be followed by other item. If you need empty markered paragraph, call Add[167] to add empty text item after the list marker. See also: rules for valid documents[74]

Return value: item index of the marker.

See also methods:

- GetListMarkerInfo[199];
- Format[182].

See also properties:

- Style[156];
- ItemCount[144].

See also methods of TCustomRichViewEdit

- ApplyListStyle[297];
- RemoveLists[334];
- ChangeListLevels[304].

See also:

- List Markers[99].

1.4.1.1.3.132 TRichView.SetPictureInfo

Changes main properties of the item of picture[90] or *hot-picture*[93] type.

```
function SetPictureInfo(ItemNo: Integer; const  AName: TRVAnsiString[623];
  Agr: TGraphic; AVAlign: TRVVAlign[648]; const ATag: TRVTag[645]): Boolean;
```

Parameters:

ItemNo – index of the item. The item must be of picture[90] or *hot-picture*[93] type (*rvsPicture or rvsHotPicture*[663]), otherwise the method raises ERichViewError[608] exception. Items are indexed from 0 to ItemCount[144]-1, GetItemStyle[194] returns type of item. Items of subdocuments (table cells[567]) are not included in the items range of the main document; for items in cells, use Cell.GetRVData[575].SetPictureInfo.

AName – name of the item, any string without line break (CR, LF) characters. It can also be set using SetItemText[232] method.

Agr – graphic object. If this item does not use a shared image (by default), **Agr** must be an unique graphic object (you cannot assign the same graphic object to two or more items) or the value returned by GetPictureInfo[201] for this item (the image is shared if *rvepShared* extra item property[627] is nonzero). If this item does not use a shared image, and **Agr** is not equal to the value returned by GetPictureInfo[201] for this item, the previously used graphic object is freed.

AVAlign – vertical alignment of the picture.

ATag – *tag* of the item.You can use value returned by GetPictureInfo[201] or GetItemTag[195] for this item. The *tag* can also be set by SetItemTag[231] method.

Return value: is reformatting needed? (are the size and the vertical align of the new picture not equal to the size and the vertical align of the old one?)

Method type: viewer-style.

Additional item properties are assigned by the methods SetItemExtraIntProperty[230] and SetItemExtraStrProperty[231] (including stretching, transparency for bitmaps, link to file name, alternative text for saving to HTML).

Unicode note: names of non-text items are ANSI strings.

Example 1

```
RichView1.SetPictureInfo(ItemNo, 'my image', Bitmap2, rvvaBaseline, 'image tag');
```

Example 2 (incorrect!)

```
bmp := TBitmap.Create;
bmp.LoadFromFile('c:\image.bmp');
RichView1.SetPictureInfo(ItemNo1, '', bmp, rvvaBaseline, '');
RichView1.SetPictureInfo(ItemNo2, '', bmp, rvvaBaseline, ''); // incorrect! you cannot as
  // the same graphic object to different items
```

Example 2 (the same, but correct)

```
bmp1 := TBitmap.Create;
bmp1.LoadFromFile('c:\image.bmp');
RichView1.SetPictureInfo(ItemNo1, '', bmp1, rvvaBaseline, '');
bmp2 := TBitmap.Create;
bmp2.Assign(bmp1);
RichView1.SetPictureInfo(ItemNo2, '', bmp2, rvvaBaseline, '');
```

Example 2 (the same, using shared images)

```
bmp := TBitmap.Create;
bmp.LoadFromFile('c:\image.bmp');
RichView1.SetPictureInfo(ItemNo1, '', bmp, rvvaBaseline, '');
RichView1.SetItemExtraIntProperty(ItemNo1, rvepShared, 1);
RichView1.SetPictureInfo(ItemNo2, '', bmp, rvvaBaseline, '');
RichView1.SetItemExtraIntProperty(ItemNo2, rvepShared, 1);
// you must free bmp yourself when RichView1 is cleared
```

See also methods:

- GetPictureInfo 201;
- GetItemStyle 194;
- Format 182;
- SetItemExtraIntProperty 230;
- SetItemExtraStrProperty 231.

See also properties:

- ItemCount 144.

See also methods of RichViewEdit:

- SetPictureInfoEd 354;
- SetCurrentPictureInfo 348.

See also:

- Modifying RichView items 51;
- Item types 29;
- *Tags* 34.

1.4.1.1.3.133 TRichView .SetSelectionBounds

Selects part of RichView document (for copying to the Clipboard)

```
procedure SetSelectionBounds(StartItemNo, StartItemOffs,
  EndItemNo, EndItemOffs: Integer);
```

Parameters

StartItemNo – index of the first selected item.

StartItemOffs:

- if the first item is a text item 88, then the selection start is before the **StartItemOffs**-th character of string (characters in strings are counted from 1, the last position (after the text item) is text length+1).

- if the first item is not a text, then
  - if **StartItemOffs**=0, then the selection start is before the first item;
  - if **StartItemOffs**=1, then the selection start is after the first item.

EndItemNo – index of the last selected item.

EndItemOffs:

- if the last item is a text item, then the selection end is before the **EndItemOffs**-th character of string (characters in strings are counted from 1, the last position (after the text item) is text length+1).
- if the last item is not a text, then
  - if **EndItemOffs**=0, then the selection end is before the last item
  - if **EndItemOffs**=1, then the selection end is after the last item.

Caret position

In TRichViewEdit[282], the caret is always at the end of the selection, so this method can be used to move caret.

Selection in table[98] cells

Before making selecting inside a table cell, activates its editing:

```
var RVData: TCustomRVFormattedData[608];

RVData := TCustomRVFormattedData[608](Table.Cells[537][r,c].Edit[610]);
RVData.SetSelectionBounds(...);
```

Deselection

There are two ways to deselect:

- to pass **StartItemNo**=-1
- to pass **StartItemNo**=**EndItemNo** and **StartItemOffs**=**EndItemOffs** (in editor, the caret is moved to the specified position).

Please be careful and do not specify incorrect values.

Additional information

This method does not repaint the component (use Invalidate)

This method generates OnSelect[265] event.

There are two useful methods to work with SetSelectionBounds: GetOffsBeforeItem[201] and GetOffsAfterItem[200].

This method must be called only when the document is formatted.

If you want to set selection bounds using richedit-like parameters (SelStart and SelLength), use RVSetSelection from RVLinear[621] unit instead.

See also methods:

- SelectAll[223];
- Deselect[181];
- GetSelectionBounds[203].

See also events:

- OnSelect[265].

See also:

- Working with selection[47];
- How to: move a caret to the beginning or to the end of document in TRichViewEdit[668];
- Example how to move caret to the beginning of paragraph[371].

See also:

- Edit[610] method of table cell.

1.4.1.1.3.134 TRichView.StartAnimation

Starts animation of pictures.

```
procedure StartAnimation;
```

(introduced in version 10)

Alternatively, animation may start automatically when document is formatted[182], if AnimationMode[135] = *rvaniOnFormat*.

See also methods:

- ResetAnimation[215];
- StopAnimation[237].

See also properties:

- AnimationMode[135].

See also:

- Animation in TRichView[56];
- Pictures[90];
- *Hot-Pictures*[93].

1.4.1.1.3.135 TRichView.StartLiveSpelling

Starts live spelling check (runs a live spelling thread).

```
procedure StartLiveSpelling;
```

(introduced in v1.9)

OnSpellingCheck[265] event must be assigned.

If live spelling thread is already active, this method clears all live spelling underlines and restarts the thread from the beginning of document.

Note: If you use RichViewActions and want to start live spelling immediately when document is loaded, call **StartLiveSpelling** in **TrvActionOpen.OnOpenFile**.

See also property of TRichViewEdit:

- LiveSpellingMode[293].

See also:

- Live spelling check in RichView[69].

1.4.1.1.3.136 TRichView.StopAnimation

Stops animation of pictures.

```
procedure StopAnimation;
```

(introduced in version 10)

Clear[175] may stop animations as well.

See also methods:

- StartAnimation[237];
- ResetAnimation[215].

See also properties:

- AnimationMode[135].

See also:

- Animation in TRichView 56;
- Pictures 90;
- *Hot-Pictures* 93.

1.4.1.1.3.137 TRichView.UpdatePaletteInfo

Call this method when screen color resolution changed (see WM_DISPLAYCHANGE window message). You need not to call this method if DoInPaletteMode 142=rvpaDoNothing.

```
procedure UpdatePaletteInfo;
```

Example 143

You also need to call UpdatePaletteInfo if you create TRichView at run-time (instead of placing at design-time on form) after you have created it.

This method is necessary only in 256-color display mode (not supported by new versions of Windows).

See also properties:

- DoInPaletteMode 142.

See also methods of TRVPrint:

- UpdatePaletteInfo 518.

1.4.1.1.4 Events

1.4.1.1.4.1 TRichView.OnAddStyle

Occurs when new style is added.

```
type
  TRVAddStyleEvent = procedure (Sender: TCustomRichView 128;
    StyleInfo: TCustomRVInfo 430) of object;

property OnAddStyle: TRVAddStyleEvent;
```

(introduced in version 11)

This event occurs when a new style is added (to TextStyles 404, ParaStyles 398, or ListStyles 396 collections of the linked TRVStyle 385 component) as a result of:

- RTF (Rich Text Format) reading (both loading and insertion)
- RVF (RichView Format) insertion;
- (if style templates are used 158) auto-correction of ParaStyleTemplateId 439 property of text styles;
- (in editor) changing the current text style on keyboard language change (see *rvoAutoSwitchLang* in EditorOptions 291);
- (in editor) ApplyStyleTemplate 299, ApplyParaStyleTemplate 298, ApplyTextStyleTemplate 301;
- (in editor, if style templates are used 158) when a user presses Enter (because of applying NextId 462 property of a style template).

Notes about additional components:

- HTML importers call this event;
- RichViewActions do not call this event, they use TRVAControlPanel.OnAddStyle instead.

1.4.1.1.4.2 TRichView.OnCheckpointVisible

Occurs when *checkpoint* (with RaiseEvent flag = *True*) becomes visible as a result of vertical scrolling.

```
type
  TRVCheckpointVisibleEvent = procedure(Sender: TCustomRichView;
    CheckpointData: TCheckpointData) of object;

property OnCheckpointVisible: TRVCheckpointVisibleEvent;
```

This event occurs if:

- RichView.CPEventKind 139 = *cpeWhenVisible* for *checkpoint* that becomes visible;
- RichView.CPEventKind = *cpeAsSectionStart* for visible *checkpoint* or the nearest *checkpoint* above visible area.

CheckpointData parameter identifies the *checkpoint*. It's possible that **CheckpointData**=*nil* (no checkpoint visible).

This event never occurs in TRichViewEdit.

See also properties:

- CPEventKind 139.

See also methods extracting information from CheckpointData:

- GetCheckpointInfo 185;
- GetCheckpointXY 186;
- GetCheckpointYEx 187;
- GetCheckpointItemNo 186;
- GetCheckpointNo 186.

See also

- Checkpoints 30.

1.4.1.1.4.3 TRichView .OnControlAction

Occurs when some important operation is performed on control inserted in RichView

```
type
  TRVControlAction =
    (rvcaAfterRVFLoad, rvcaDestroy,
     rvcaMoveToUndoList, rvcaMoveFromUndoList,
     rvcaDestroyInUndoList, rvcaBeforeRVFSave,
     rvcaAfterRVFSave);

  TRVControlActionEvent =
    procedure (Sender: TCustomRichView;
      ControlAction: TRVControlAction;
      ItemNo: Integer;
      var ctrl: TControl) of object;

property OnControlAction: TRVControlActionEvent;
```

(Introduced in version 1.3, changed in v1.4 and v10)

Warning: this event can occur from RichView destructor (when clearing document before destruction). Be careful if you access other controls from this event: they may be already destroyed!

This event occurs when some changes were done (or are being done) on inserted control **ctrl**.

Position of this control in the document is defined by the "hidden" parameter **Sender**.Style 156.RVData (must be typecasted to TCustomRVData 608) and **ItemNo** parameter. **ItemNo** is an index of item (containing **ctrl**) in the document **Sender**.Style.RVData (it may be RVData 151 of the main RichView (**Sender**), cell 567, or RVData 151 of cell inplace editor).

ControlAction	Meaning
rvcaAfterRVFLoad	The control was created and loaded from RVF (file, stream, database field or Clipboard);
rvcaDestroy	The control will be destroyed soon (removing items

	from RichView, or destroying RichView itself)
rvcaMoveToUndoList	The control was moved to undo/redo buffer. It will not be displayed, but it will not be destroyed yet.
rvcaMoveFromUndoList	The control was moved from undo/redo list back to the editor.
rvcaDestroyInUndoList	The control will be destroyed together with undo/redo buffer (undo item); this control is already not in the editor; only **ctrl** parameter is valid in this case, **ItemNo**=-1;
rvcaBeforeRVFSave	Before saving to RVF
rvcaAfterRVFSave	After saving to RVF

Note: do not perform any changes on RichView when processing this event.

Note: TRVControlAction is defined in RVStyle unit.

See also:

- OnItemAction[244].

1.4.1.1.4.4 TRichView.OnCopy

Occurs when copying to the Clipboard

```
property OnCopy: TNotifyEvent;
```

(Introduced in version 1.4)

Occurs inside CopyDef[177] and Copy[177] methods, or when copying from keyboard.

This event allows copying to the Clipboard in your own formats (occurs between Clipboard.Open and Clipboard.Close).

See also:

- TRichViewEdit.OnPaste[365];
- Working with Clipboard[48].

1.4.1.1.4.5 TRichView.OnGetItemCursor

Allows defining custom custom cursors for items.

```
type
  TRVGetItemCursorEvent = procedure (Sender: TCustomRichView[128];
    RVData: TCustomRVData[608]; ItemNo: Integer; var Cursor: TCursor) of object;

property OnGetItemCursor: TRVGetItemCursorEvent;
```

(introduced in version 12)

This event occurs when mouse pointer moves over the item.

Input Parameters

RVData – document containing this item (RichView.RVData[151], or table cell, or cell inplace editor's RVData[151])

ItemNo – index of the item inside **RVData**.

Output Parameter

Cursor – cursor to display.

See also:
- OnItemHint 246.

1.4.1.1.4.6 TRichView .OnHTMLSaveImage

Occurs when TRichView saves item containing image to HTML file or stream. This event allows to modify saving procedure for all (or some) images.

```
type
  TRVHTMLSaveImageEvent =
    procedure (Sender: TCustomRichView; RVData: TCustomRVData;
      ItemNo: Integer; const Path: String;
      BackgroundColor: TColor; var Location: String;
      var DoDefault: Boolean) of object;

property OnHTMLSaveImage: TRVHTMLSaveImageEvent;
```

(introduced in version 1.4)

This event is called for the following item types:
- pictures 90, *hot-pictures* 93,
- *bullets* 96, *hotspots* 97,
- list markers 99 (paragraph bullets) with images,
- tables 98 (saving background image 530),
- for background image 136 and cells' background images 567.

There is more easy-to-use event: OnSaveImage2 260.

By default:

Images are saved in separate files as Jpegs (or as bitmaps for Delphi2, or if RVDONOTUSEJPEGIMAGE is defined) and have names composed as ImagesPrefix + Number + '.jpg', where
- ImagesPrefix – parameter of SaveHTML 216/SaveHTMLEx 217;
- Number – some number making this file name unique (image files can override existing ones or not, depending on parameters of SaveHTML/SaveHTMLEx). *Bullets* and *hotspots* items having the same image list, image index and background color are saved in the same file (the same is for list markers).

Transparent color of *bullets*, *hotspots* and transparent image formats (TIcon, TMetafile, gifs) is replaced with background color (which can be color of component (TRichView.Color 139/TRVStyle.Color 387), color of cell (TRVTableCellData.Color 570) or color of paragraph background (TRVBackroundRect.Color 467).

You can define formats that must not be converted to Jpeg 672.

Input Parameters

RVData – document containing item that is being saved to image; it can be **Sender**.RVData 151, table cell 567, or RVData of cell inplace-editor.

ItemNo – index of this item inside **RVData**. If this event is called for background image (document or cell), **ItemNo**=-1.

Path – destination directory of HTML file (or Path parameter of SaveHTMLToStream 218/SaveHTMLToStreamEx 218).

BackgroundColor – color of background below the item (can be ignored, if you can preserve transparency of image)

Location – if *rvsoUseItemImageFileNames* is included in the Options parameter of SaveHTML 216/SaveHTMLEx 217, this is a path to the image defined in the *rvespImageFileName* item property 630, relative to **Path**. Otherwise, initial value of this parameter is empty.

Output Parameters

Location – set this parameter to file name of image where you saved it. This string will be inserted in HTML file.

DoDefault – set to *False* if you saved this item as image yourself, to disallow default saving. If **DoDefault** is *True*, then after calling this event:

- if *rvsoUseItemImageFileNames* is included in the Options parameter of SaveHTML 216/SaveHTMLEx 217 and value of *rvespImageFileName* item property 630 is not empty, this value is written in HTML (relative to **Path**); otherwise
- OnSaveImage2 260 is called (if its handler is assigned); if it is not assigned or OnSaveImage2's DoDefault parameter is *True*, the default image saving procedure is called.

Inside this event you can use:

- **Sender**.imgSaveNo – value which was used as Number for file name for the previously saved image;
- **RVData**.GetNextFileName – function returning image file name with the given image prefix and extension:

```
function GetNextFileName(const ImagesPrefix, Path, Ext: String;
  var imgSaveNo: Integer; OverrideFiles: Boolean): String;
```

where

ImagesPrefix – the first part of image file name (for example, you can set it to ImagesPrefix parameter of SaveHTML 216/SaveHTMLEx 217);

Path – path for the image (for example, you can use **Path** parameter of this event);

Ext – file extension, for example '.jpg' (you can use GraphicExtension(TGraphicClass(Graphic.ClassType)));

imgSaveNo – this function will use this value incremented one or several times to compose a name of image; returns the incremented value;

OverrideFiles – if set to *False*, this function will check existing files and returns an unique file name (by incrementing **imgSaveNo** several times)

Return value: full file name of the image. You can use it to save image and assign it to **Location** parameter of the event (this file name includes path, use ExtractFileName or ExtractRelativePath).

Alternatively, you can use SavePicture 219.

See also events:

- OnSaveImage2 260;
- OnSaveItemToFile 261.

See also

- Export to HTML 62.

1.4.1.1.4.7 TRichView .OnImportPicture

Occurs when loading RTF file or stream containing links to image files. This event also can be used by ***TRVHtmlImporter***.

```
type
  TRVImportPictureEvent =
    procedure (Sender: TCustomRichView;
    const Location: String; Width, Height: Integer;
    var Graphic: TGraphic) of object;

property OnImportPicture: TRVImportPictureEvent;
```

(introduced in version 1.8)

Usually pictures are saved inside RTF file/stream. But sometimes RTF contains links to external image files (file names or URLs).

If these links are local (like 'c:\images\image.bmp'), TRichView can load these files itself. But if images are on

the Internet (like 'http://www.trichview.com/images/headers/what_is.gif'), it makes sense to process this event and to download the image.

Input parameters

Location – path to the file (local or URL);

Width, Height – reserved, equal to 0;

Graphic – *nil.*

Output parameters

Graphic – set it to the loaded image. If you leave it equal to *nil*, TRichView will attempt to load the image itself.

(TRichView can load image file only if its extension is associated with some graphic class. Delphi does it automatically for TBitmap, TIcon, TMetafile, TJpegImage. If you use third-party graphic classes [671], you can register them using TPicture.RegisterFileFormat class procedure).

Example: downloading images from http location using Indy (IdHTTP1: TIdHTTP)

Unfortunately, Delphi does not have methods returning graphic class by file extension. So we need to create a temporal file, save image there, then load it in TPicture.

```
uses RVFuncs;

{ Returns the system directory for temporal files }
function GetTempDir: String;
var l: Integer;
begin
  // Warning: : The GetTempPath function
  // does not verify that the returned
  // directory exists.
  SetLength(Result, 300);
  l := GetTempPath(300, PChar(Result));
  SetLength(Result, l);
end;

{ Returns name for temporal file with the
  given extension }
function GetTempFileName(const Ext: String): String;
var Path: String;
    v: Integer;
begin
  Path := GetTempDir+'tmp';
  v := Random(MaxInt div 2);
  repeat
    inc(v);
    Result := Path+IntToStr(v)+Ext;
  until not FileExists(Result);
end;

{ TMyRichView.OnImportPicture }
procedure TMyForm.MyRichViewImportPicture(Sender: TCustomRichView;
  const Location: String; Width, Height: Integer;
  var Graphic: TGraphic);
var Stream: TMemoryStream;
    FileStream: TFileStream;
    FileName: String;
    pic: TPicture;
```

```
begin
  if Pos('http://', LowerCase(Location))=1 then
  begin
    Stream := TMemoryStream.Create;
    try
      // 1. downloading
      IdHTTP1.Get(Location, Stream);
      // 2. creating temporal file
      FileName := GetTempFileName(ExtractFileExt(Location));
      FileStream := TFileStream.Create(FileName, fmCreate);
      try
        FileStream.CopyFrom(Stream, 0);
      finally
        FileStream.Free;
      end;
      // 3. reading and deleting temporal file
      pic := TPicture.Create;
      try
        pic.LoadFromFile(FileName);
        Graphic := RV_CreateGraphics(TGraphicClass(pic.Graphic.ClassType));
        Graphic.Assign(pic.Graphic);
      finally
        pic.Free;
        DeleteFile(FileName);
      end;
    except
      Graphic.Free;
      Graphic := nil;
    end;
    Stream.Free;
  end;
end;
```

See also methods of TRichView:

- LoadRTF 209;
- LoadRTFFromStream 210.

See also methods of TRichViewEdit:

- InsertRTFFromFileEd 323;
- InsertRTFFromStreamEd 323.

1.4.1.1.4.8 TRichView .OnItemAction

Occurs on insertion, deletion and some other modifications of item

```
type
  TRVItemAction = (rviaInserting, rviaInserted,
    rviaTextModifying, rviaDestroying, rviaMovingToUndoList)

  TRVItemActionEvent = procedure (Sender: TCustomRichView 128;
    ItemAction: TRVItemAction; Item: TCustomRVItemInfo 526;
    var Text: TRVRawByteString 638; RVData: TCustomRVData 608) of object;

property OnItemAction: TRVItemActionEvent;
```

(introduced in version 1.7)

Parameters

Item – item[526] for which this event is generated.

Text – text associated with this item. If this is a text item, this text is displayed in RichView.

RVData – document containing Item (it can be **Sender**.RVData[151], table cell[567], or RVData of cell inplace-editor).

ItemAction identifies operation that calls this event.

ItemAction	Meaning
rviaInserting	The event occurs before the item is inserted in **Sender** (in **RVData**). You can modify **Text**. This event also occurs when moving items to cell inplace editor and back to the cell.
rviaInserted	The event occurs after the item is inserted in the **Sender** (in **RVData**). Ignore **Text** parameter. This event also occurs when moving items to cell inplace editor and back to the cell.
rviaMovingToUndoList	The event occurs when item is moved (from **RVData**) to undo list. Ignore **Text** parameter.
rviaDestroying	Occurs when item is destroyed. Ignore **Text** parameter. If the item is in undo/redo list, **RVData** is *nil*. Otherwise, this is an object containing this item.
rviaTextModifying	Occurs when item text (**Text**) is modified as a result of editing operation. Be careful when modifying text. It's highly not recommended to do any modification of **Text** except for replacing one characters to others.

Example (converting all non-Unicode text to upper case):

```
procedure TForm1.RichViewEdit1ItemAction(Sender: TCustomRichView;
  ItemAction: TRVItemAction; Item: TCustomRVItemInfo;
  var Text: TRVRawByteString[638]; RVData: TCustomRVData);
begin
  if (ItemAction in [rviaInserting, rviaTextModifying]) and
     (Item.StyleNo>=0) and
     not (rvioUnicode in Item.ItemOptions) then
    Text := AnsiUpperCase(Text);
end;
```

Warning: this event can occur from RichView destructor (when clearing document before destruction). Be careful if you access other controls from this event, they may be already destroyed.

See also events:

- OnControlAction[239].

See also methods:

- GetItem[190];
- GetItemNo[194].

1.4.1.1.4.9 TRichView.OnItemHint

Allows to define custom popup hints for items.

```
type
  TRVItemHintEvent = procedure (Sender: TCustomRichView[128];
    RVData: TCustomRVData[608]; ItemNo: Integer;
    var HintText: String) of object;

property OnItemHint: TRVItemHintEvent;
```

This event occurs when mouse pointer moves over the item.

Parameters

RVData – document containing this item (RichView.RVData[151], or table cell, or cell inplace editor's RVData[151])

ItemNo – index of the item inside **RVData**.

HintText – on input, it's a value of *rvespHint*[630] item property[630]. On output, it's a message that will be displayed.

Unicode note: hints are Unicode strings for Delphi/C++Builder 2009, and ANSI strings for older versions of Delphi/C++Builder.

Item popup hint can be displayed if *rvoShowItemHints* is in RichView.Options[146] and RichView.ShowHint=*True*.

Cell Hint[571]s cannot be altered in this event.

See also:

- OnGetItemCursor[240].

Example

Besides the default hint, this code displays:

- for hyperlinks: tag;
- for images: Delphi class type, width and height.

```
procedure TMyForm.MyRichViewEditItemHint(Sender: TCustomRichView[128];
  RVData: TCustomRVData[608]; ItemNo: Integer; var HintText: String);
var Tag: TRVTag[645];
    VAlign: TRVVAlign[648];
    Name: TRVAnsiString[623];
    gr: TGraphic;
begin
  if RVData.GetItem[190](ItemNo).GetBoolValueEx(rvbpJump, Sender.Style[156]) then begin
    if HintText<>'' then
      HintText := HintText+#13;
    HintText := HintText + 'Target: '+RVData.GetItemTag[195](ItemNo);
  end;
  if (RVData.GetItemStyle[194](ItemNo)=rvsPicture[663]) or
     (RVData.GetItemStyle[194](ItemNo)=rvsHotPicture[663]) then begin
    if HintText<>'' then
      HintText := HintText+#13;
    RVData.GetPictureInfo[201](ItemNo, Name, gr, VAlign, Tag);
    HintText := HintText+gr.ClassName+' ('+IntToStr(gr.Width)+','+IntToStr(gr.Height)+')';
  end;
end;
```

1.4.1.1.4.10 TRichView.OnJump

Occurs when user clicks on hypertext item.

```
type
  TJumpEvent = procedure(Sender: TObject; id: Integer) of object;

property OnJump: TJumpEvent;
```

Parameter

id is an identifier of hypertext link (the first link in the document has id=FirstJumpNo[144], the next link has **id**=FirstJumpNo[144]+1, and so on).

In editor, this numeration is available only when the editor works in hypertext mode:

- while user holds **Ctrl** key (see EditorOptions[291]) until he/she releases it, or the document is modified, or the editor loses focus;
- if ReadOnly[294]=True.

So, when using this event in editor, perform all actions that use this **id** before any action changing document or moving focus out of editor.

In viewer, this even can also be initiated using keyboard, if TabNavigation[157]<>*rvtnNone*. User uses **Tab** to highlight the hyperlink, then **Enter** to activate it (to call this event).

Example:

```
procedure TMyForm.MyRichViewJump(Sender: TObject; id: Integer);
var ItemNo: Integer;
    RVData: TCustomRVFormattedData[608];
    s: String;
begin
  MyRichView.GetJumpPointLocation[197](id, RVData, ItemNo);
  s := RVData.GetItemTag[195](ItemNo);
  Application.MessageBox(PChar(s), 'Link', MB_OK or MB_ICONINFORMATION);
end;
```

See also events:

- OnRVMouseMove[254].

See also:

- RichView hypertext[35].

1.4.1.1.4.11 TRichView.OnPaint

Allows painting over the richview document

```
type
  TRVPaintEvent = procedure (Sender: TCustomRichView;
    Canvas: TCanvas; Prepaint: Boolean) of object;

property OnPaint: TRVPaintEvent;
```

(introduced in version 1.7)

Parameters:

Canvas – canvas for painting.

Prepaint – reserved for future use.This event is called when all standard painting is done.

This event does not affect printing (see OnPagePrepaint[495], OnPagePostpaint[495]).

Demo project:

- **Demos\Delphi\Assorted\Custom Draw\CustomDraw**
- **Demos\CBuilder\Assorted\Custom Draw\CustomDraw**

1.4.1.1.4.12 TRichView .OnProgress

Occurs on long operations

```
type
  TRVLongOperation = ( // defined in RVStyle unit
    rvloRTFRead, rvloRTFWrite,
    rvloRVFRead, rvloRVFWrite,
    rvloHTMLRead, rvloHTMLWrite,
    rvloTextRead, rvloTextWrite,
    rvloConvertImport, rvloConvertExport,
    rvloXMLRead, rvloXMLWrite,
    rvloDocXRead, rvloDocXWrite,
    rvloOtherRead, rvloOtherWrite
    );
  TRVProgressStage = // defined in RVStyle unit
    (rvpstgStarting, rvpstgRunning, rvpstgEnding);
type
  TRVProgressEvent = procedure (Sender: TCustomRichView;
    Operation: TRVLongOperation; Stage: TRVProgressStage;
    PercentDone: Byte) of object;

property OnProgress: TRVProgressEvent;
```

(introduced in v1.9, changed in v14)

First, it is called with parameter **Stage**=*rvpstgStarting*.

Next, it is called several times with parameter **Stage**=*rvpstgRunning*, with **PercentDone** in the range 0..100.

Finally, it is called with parameter **Stage**=*rvpstgEnding*.

This event occurs on the following operations:

- reading/writing RTF;
- reading/writing RVF [59];
- writing HTML;
- writing DocX;
- reading/writing a plain ANSI or Unicode text;
- when calling TRVOfficeConverter.ImportRV [502] and ExportRV [500].

Note: the event is called if the document/source file is large enough.

See also:

- TRVOfficeConverter.OnConverting [503]

1.4.1.1.4.13 TRichView .OnReadHyperlink

Occurs when reading hyperlinks (for example, from RTF file)

```
type
  TRVLoadFormat =
    (rvlfText,rvlfHTML,rvlfRTF,rvlfRVF,rvlfURL);

  TRVReadHyperlink =
    procedure (Sender: TCustomRichView;
      const Target, Extras: String;
```

```
      DocFormat: TRVLoadFormat;
     var StyleNo: Integer; var ItemTag: TRVTag[645];
     var ItemName: TRVRawByteString[638]) of object;

property OnReadHyperlink: TRVReadHyperlink;
```

By default, hyperlink targets is saved in the item tag[34]. Use this event to customize loading.

Input parameters:

Target – target of hypertext link (for example, an Internet address, a file name, or a bookmark/anchor).

Extras – additional information stored with link. For example, if **Extras**='\o"hint_text"', then **Target** is read from RTF and the link has a popup hint 'hint_text'.

TRichView automatically writes items hints in *rvespHint* item's extra string property[630], and adds '#' to the beginning of links to bookmarks.

DocFormat – format of loaded document.

DocFormat	Meaning
rvlfRTF	The event occurs when reading RTF files containing hyperlinks (RTF loading, pasting, or accepting on drag&drop).
rvlfURL	The event occurs: 1) on drag&drop, when accepting URL dropped to TRichViewEdit[282] (*rvddURL* must be in AcceptDragDropFormats[287]). 2) when pasting from the Clipboard, when pasting URL to TRichViewEdit[282] (*rvddURL* must be in AccepPasteFormats[287]).
rvlfHTML	The event occurs when reading HTML files using ***TRVHtmlImporter***, if *rvhtmloAutoHyperlinks* is excluded from RVHtmlImporter.Options. Or the event occurs when reading HTML files using ***TRVHtmlViewImporter***, if RVHtmlViewImporter.AutoHyperlinks=*False*.

StyleNo:

- if the item is a picture, it is initially equal to *rvsHotPicture*. You can change it to *rvsPicture*, if you want to disable this link.
- if the item is a text, it is index of style[404] of this text. You can modify it to point to another (hypertext[445]) style. If **DocFormat**=*rvlfURL*, the initial style may be not hypertext, so you need to provide the index of hypertext style in this parameter.

ItemName:

- if the item is a picture, it is initially equal to '' (empty string) . You can set it to value which will be assigned to name of this item.
- if the item is a text, it is initially a text to display. You can modify it.

Output parameters:

StyleNo – style that will be used for the loaded item. See comments for input parameters.

ItemName – value that will be assigned to item text. See comments for input parameters.

ItemTag – value that will be assigned to tag[34] of this item. A good place to store hyperlink target.

Example 1

This example does the same work as it the event is not assigned: reading the link target in the item tag [34].

```
procedure TMyForm.MyRichViewReadHyperlink(Sender: TCustomRichView;
  const Target, Extras: String; DocFormat: TRVLoadFormat;
  var StyleNo: Integer; var ItemTag: TRVTag[645];
  var ItemName: TRVRawByteString[638]);
begin
  ItemTag := Target;
end;
```

More complicated example

This example assumes that you included *rvddURL* in AcceptDragDropFormat [287] and want to accept hyperlinks. Besides, it's assumed that "Allow adding styles dynamically" is set in the component editor [134] (or properties are set to the proper values to allow saving the changed collection of text styles).
This code uses blue underlined font for hyperlinks.

```
procedure TMyForm.MyRichViewReadHyperlink(Sender: TCustomRichView;
  const Target, Extras: String; DocFormat: TRVLoadFormat;
  var StyleNo: Integer; var ItemTag: TRVTag[645];
  var ItemName: TRVRawByteString[638]);
var FontStyle: TFontInfo[444];
begin
  ItemTag := Target;
  if DocFormat=rvddURL then
  begin
    FontStyle := TFontInfo[444].Create(nil);
    FontStyle.Assign(Sender.Style[156].TextStyles[404][StyleNo]);
    FontStyle.Color[435] := clBlue;
    FontStyle.Style[441] := FontStyle.Style + [fsUnderline];
    FontStyle.Jump[445] := True;
    StyleNo := Sender.Style.FindTextStyle[407](FontStyle);
    FontStyle.Free;
  end;
end;
```

The same example using RichViewActions:

```
procedure TMyForm.MyRichViewReadHyperlink(Sender: TCustomRichView;
  const Target, Extras: String; DocFormat: TRVLoadFormat;
  var StyleNo: Integer; var ItemTag: TRVTag[645];
  var ItemName: TRVRawByteString[638]);
begin
  ItemTag := Target;
  if DocFormat=rvddURL then
    StyleNo := rvActionInsertHyperlink1.GetHyperlinkStyleNo(Sender as TRichViewEdit);
end;
```

See also events:

- OnWriteHyperlink [267].

See also properties of RVRTFReadProperties [151]:

- BasePathLinks [269].

1.4.1.1.4.14 TRichView.OnRVDblClick

Occurs when user double-clicks the primary mouse button when the mouse pointer is above the item in RichView (or on single left-click, if *rvsSingleClick* is in Options [146])

```
type
```

```
  TRVDblClickEvent =
    procedure(Sender: TCustomRichView[128]; ClickedWord: TRVRawByteString[638];
      Style: Integer) of object;

property OnRVDblClick:TRVDblClickEvent;
```

Parameters

Style – style (type) of the clicked item.

If **Style**<0 then the clicked item is not a text item, and **Style** is a type of this item (see Item Types[29]). In this case **ClickedWord** contains a name of this item, ANSI string.

If **Style**>=0, then this is a text item[88], and **Style** is an index in the collection of text styles (RichView.Style.TextStyles[404]). In this case **ClickedWord** is a word below the mouse pointer. "Word" is defined as a part of string between delimiters. Delimiters are listed in Delimiters[140] property. For Unicode[446] text items, **ClickedWord** contains "raw Unicode". For non-Unicode text items, **ClickedWord** contains ANSI string.

This event is not generated when the user clicks not on item (on background)

See also events:

- OnRVRightClick[255].

See also properties:

- Delimiters[140].

1.4.1.1.4.15 TRichView .OnRVFControlNeeded

Occurs when reading inserted control[94] from RVF file or stream, and this control was saved without its "body".

```
type
  TRVFControlNeededEvent =
    procedure(Sender: TCustomRichView[128]; Name: String;
      const Tag: TRVTag[645]; var ctrl: TControl) of object;

property OnRVFControlNeeded: TRVFControlNeededEvent;
```

By default, values of all published properties of controls are saved in RVF documents.

If you exclude *rvfoSaveControlsBody* from RichView.RVFOptions[152], none of control properties are saved. Only the item name (value returned by GetItemText[195]) and *tag* (value returned by GetItemTag[195]) are saved. These values allow to identify the control on loading. When reading such RVF files, OnRVFControlNeeded is called for each control.

Input parameters:

Name – name of the control's item (this is a value returned by GetItemText[195] (converted to Unicode for Delphi/C++Builder 2009 or newer); do not confuse with Name property of controls, see Item types[29]).

Tag – *tag* of the control's item (this is a value returned by GetItemTag[195]; do not confuse with Tag property of controls, see "Tags"[34]).

Output parameter:

ctrl – control to insert in document. Create this control in this event. Do not destroy it (RichView will destroy it when necessary).

See also:

- RVF overview[59], contains example.

1.4.1.1.4.16 TRichView.OnRVFImageListNeeded

Occurs when reading *bullet*[96], *hotspot*[97] or list level with imagelist picture[475] from RVF.

```
type
  TRVFImageListNeededEvent =
    procedure(Sender: TCustomRichView[128];
      ImageListTag: Integer;
      var il: TCustomImageList) of object;

property OnRVFImageListNeeded: TRVFImageListNeededEvent;
```

If you do not process this event, RichView is not able to read *bullets*[96], *hotspots*[97] and list level with imagelist from RVF file or stream.

When saving these items, RichView saves Tag of the corresponding ImageList (do not confuse with RichView items' tags[34], this is TCustomImageList.Tag property). This value is passed as **ImageListTag** parameter of this event on reading.

Assign your ImageList to **il** parameter (note: RichView does not own ImageLists, it will not destroy it).

This event is generated for each *bullet*, *hotspot* and list levels with imagelist in RVF, so it is not recommended to perform time consuming calculations in this event.

See also:

- RVF overview[59], contains example.

1.4.1.1.4.17 TRichView.OnRVFPictureNeeded

Occurs when reading picture[90] or *hot-picture*[93] from RVF file or stream, and this picture was saved without its "body"

```
type
  TRVFPictureNeededEvent = procedure (Sender: TCustomRichView[128];
    const ItemName: String; Item: TRVNonTextItemInfo[526];
    Index1, Index2: Integer; var gr: TGraphic) of object;

property OnRVFPictureNeeded: TRVFPictureNeededEvent;
```

(the parameters are changed in version 14)

By default, images are stored inside RVF documents.

If you exclude *rvfoSavePicturesBody* from RichView.RVFOptions[152], images are not saved. Only item name (value returned by GetItemText[195]) and *tag*[34] (value returned by GetItemTag[195]) are saved. These values allow to identify the image on loading. When reading such RVF files, OnRVFPictureNeeded is called for each image.

This event is called for the following objects:

- pictures[90] and hot-pictures[93];
- table[98] background images[530] (even if the saved table did not have a background image initially)
- table cell background images[567] (for each table cell, even for cells that did not have a background image initially)

Input parameters:

ItemName – the name of the item (this is a value returned by GetItemText[195]) (converted to Unicode for Delphi/C++Builder 2009 or newer). This parameter is empty when the event is called for a table cell.

Item – the item that will use this image. This is an object of one of the following types: TRVGraphicItemInfo,

TRVHotGraphicItemInfo, TRVTableItemInfo[526].

Index1, Index2 – integer values allowing to identify image inside the item. When called for a picture, a hot-picture, or a table, they are (-1, -1). When called for a table cell, Index1 contains the row index, Index2 contains the column index.

Output parameter:

gr – image (for example, of TBitmap type) to insert in the document. Create it in this event. Do not destroy it (RichView will destroy it when necessary)

The following values may be useful:

- Item.Tag
- if Item is TRVGraphicItemInfo: TRVGraphicItemInfo(Item).ImageWidth and TRVGraphicItemInfo(Item).ImageHeight (a size of a stretched picture)
- if Item is TRVGraphicItemInfo: TRVGraphicItemInfo(Item).ImageFileName
- if Item is TRVTableItemInfo: TRVTableItemInfo(Item).BackgroundImageFileName

See also:

- RVF overview[59], contains an example.

1.4.1.1.4.18 TRichView.OnRVMouseDown

Occurs when the user presses a mouse button with the mouse pointer over the control.

```
type
  TRVMouseEvent =
    procedure(Sender: TCustomRichView[128]; Button: TMouseButton;
    Shift: TShiftState; ItemNo, X, Y: Integer) of object;

property OnRVMouseUp: TRVMouseEvent;
```

Similar to OnMouseDown, but you also have information about index of item under the mouse pointer.

ItemNo is an index of this item, or -1 if there is no item at the position of the mouse pointer.

The OnRVMouseDown event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. **X** and **Y** are the pixel coordinates of the mouse pointer in the client area of the **Sender**.

When clicking on table, this event returns **ItemNo** of this table in the root document, even if user clicked on item inside this table.

Use the following procedure if you want to know the exact clicked item:

```
procedure TForm1.MyRichViewRVMouseDown(Sender: TCustomRichView[128];
  Button: TMouseButton; Shift: TShiftState;
  ItemNo, X, Y: Integer);
var LRVData: TCustomRVFormattedData[608];
    LItemNo, LOffs: Integer;
    pt: TPoint;
begin
  pt := MyRichView.ClientToDocument[176](Point(X,Y));
  if not MyRichView.GetItemAt[190](pt.X, pt.Y, LRVData, LItemNo, LOffs, True) then
    exit;
  // item is the LItemNo-th item in LRVData
  // example of using: LRVData.GetItemStyle[194] (LItemNo)
  ..
end;
```

Note: if you want to create code responding on clicking on item, it's recommended to use OnRVMouseUp[254]

instead of OnRVMouseDown.

See also events:

- OnRVMouseUp 254.

1.4.1.1.4.19 TRichView.OnRVMouseMove

Occurs when mouse pointer is moved to the area of hypertext link.

```
type
  TRVMouseMoveEvent = procedure(Sender: TObject; id: Integer) of object

property OnRVMouseMove: TRVMouseMoveEvent
```

id is an identifier of hypertext link (the first link in the document has **id**=FirstJumpNo 144, the next link has id=FirstJumpNo 144+1, and so on).

In editor, this numeration is available only when the editor works in hypertext mode:

- while user holds Ctrl key (see EditorOptions 291) until he/she releases it, or the document is modified, or the editor loses focus;
- if ReadOnly 294=True.

So, when using this event in editor, perform all actions that use this **id** before any action changing document or moving focus out of editor.

If **id**=-1, the mouse pointer is not above hyperlink.

This event is never called two times in succession for the same hypertext link.

See also events:

- OnJump 247.

See also:

- RichView hypertext overview 35, contains example.

1.4.1.1.4.20 TRichView.OnRVMouseUp

Occurs when the user releases a mouse button that was pressed with the mouse pointer over the component.

```
type
  TRVMouseEvent = Procedure(Sender: TCustomRichView 128;
    Button: TMouseButton; Shift: TShiftState;
    ItemNo, X, Y: Integer) of object;

property OnRVMouseUp: TRVMouseEvent;
```

Similar to OnMouseUp, but you also have information about index of item under the mouse pointer.

ItemNo is an index of this item, or -1 if there is no item at the position of the mouse pointer.

The OnRVMouseUp event handler can respond to left, right, or center mouse button presses and shift key plus mouse-button combinations. Shift keys are the Shift, Ctrl, and Alt keys. **X** and **Y** are the pixel coordinates of the mouse pointer in the client area of the **Sender**.

When clicking on table, this event returns **ItemNo** of this table in the root document, even if user clicked on item inside this table.

Use the following procedure if you want to know the exact clicked item:

```
procedure TForm1.MyRichViewMouseUp(Sender: TCustomRichView 128;
  Button: TMouseButton; Shift: TShiftState;
  ItemNo, X, Y: Integer);
```

```
var LRVData: TCustomRVFormattedData[608];
    LItemNo, LOffs: Integer;
    pt: TPoint;
begin
  pt := MyRichView.ClientToDocument[176](Point(X,Y));
  if not MyRichView.GetItemAt[190](pt.X, pt.Y, LRVData, LItemNo, LOffs, True) then
    exit;
  // item is the LItemNo-th item in LRVData
  // example of using: LRVData.GetItemStyle[194] (LItemNo)
  ..
end;
```

See also events:

- OnRVMouseDown[253];
- OnRVRightClick[255].

1.4.1.1.4.21 TRichView.OnRVRightClick

Occurs when the user releases the right mouse button over the RichView component.

```
type
  TRVRightClickEvent = procedure(Sender: TCustomRichView[128];
    ClickedWord: TRVRawByteString[638]; Style, X, Y: Integer) of object;

property OnRVRightClick:TRVRightClickEvent;
```

This event is almost obsolete. There are more convenient events:

- OnRVMouseDown[253],
- OnRVMouseUp[254].

OnRVRightClick event occurs if user releases right mouse button over some item of RichView.

If **Style**<0 then this item is not a text item, and **Style** is a type of this item (such as *rvsPicture*[663], *rvsBreak*[663], *rvsHotspot*[663] or *rvsBullet*[663]). In this case **ClickedWord** contains name of this item (see Item types[29]), ANSI string.

If **Style**>=0, this is a text item, and **Style** is an index in the collection of text styles (RichView.Style.TextStyles[404]). In this case **ClickedWord** is a word which user has clicked. "Word" is defined as a part of string between delimiters. Delimiters are listed in Delimiters[140] property. For Unicode[446] text items, **ClickedWord** contains "raw Unicode". For non-Unicode text items, **ClickedWord** contains ANSI string.

X,**Y** are screen coordinates of mouse pointer (more convenient for displaying popup menu than client coordinates)

This event has the following limitations (which do not exist in OnRVMouseDown[253], OnRVMouseUp[254]):

- you can not get further information about the clicked item (you do not know index of this item);
- this event is not generated when user clicks not on item (on background).
-

See also events:

- OnRVDblClick[250].

See also properties:

- Delimiters[140].

1.4.1.1.4.22 TRichView .OnSaveComponentToFile

Occurs when RichView wants to save inserted control [94] in text, RTF, HTML, or DocX file (or stream).

```
type
  TRVSaveComponentToFileEvent = procedure(
    Sender: TCustomRichView[128]; Path: String;
    SaveMe: TPersistent; SaveFormat: TRVSaveFormat[639];
    var OutStr:String) of object;

property OnSaveComponentToFile: TRVSaveComponentToFileEvent;
```

By default, inserted controls are not saved in HTML, RTF, DocX and text files. But you can use this event to save them.

Input parameters:

SaveMe – component to save.

SaveFormat identifies file format, one of *rvsfText*, *rvsfHTML*, *rvsfRTF, rvsfDocX*.

Path – path where the file is saved. May be used, for example, for saving images (see SavePicture [219] method).

Output parameter:

OutStr – text to insert in the output file.

For Delphi/C++Builder 2009 or newer: **OutStr** is a Unicode string. For the older versions of Delphi/C++Builder **OutStr** is an ANSI string.

For DocX, **OutStr** is inserted inside a "run", i.e. between <w:r> and </w:r>.

Example:

```
procedure TMyForm.MyRichViewSaveComponentToFile(
  Sender: TCustomRichView; Path: String;
  SaveMe: TPersistent; SaveFormat: TRVSaveFormat;
  var OutStr: String);
var bmp: TBitmap;
  ImageFileName: String;
begin
 case SaveFormat of
   rvsfText:
     begin
       if SaveMe is TButton then
         OutStr := '['+TButton(SaveMe).Caption+']';
       else if SaveMe is TImage then
         OutStr := '<Image of '+TImage(SaveMe).Hint+'>';
      end;
   rvsfHTML:
     begin
       if SaveMe is TButton then
         OutStr := '<FORM><INPUT type="button"' +
                   'value="' + TButton(SaveMe).Caption +
                   '" onClick="alert(''Test'')"></FORM>';
       else if SaveMe is TImage then begin
         bmp         := TBitmap.Create;
         bmp.Height  := TImage(SaveMe).Height;
         bmp.Width   := TImage(SaveMe).Width;
         bmp.Canvas.Draw(0,0, TImage(SaveMe).Picture.Bitmap);
         ImageFileName := MyRichView.SavePicture(SaveFormat,Path,bmp);
         OutStr      := '<IMG src="'+ImageFileName+'" alt='+
                        TImage(SaveMe).Hint+'>';
```

```
          bmp.Free;
        end;
      end;
    rvsfRTF:
      begin
        OutStr := '\plain\b ('+SaveMe.ClassName+')';
      end;
 end;
end;
```

This code saves

- TButton controls in text file as "[Button.Caption]" string;
- TButton controls in HTML file as form with button;
- TImage controls in text file as "<Image of Image.Hint>" string;
- TImage controls in HTML file as image (using SavePicture) with alternative text "Image.Hint";
- All controls in RTF as "(Class of control)".

See also:

- OnSaveItemToFile [261];
- Saving and loading [58];
- Export to HTML [62].

1.4.1.1.4.23 TRichView.OnSaveDocXExtra

Allows saving additional information in DocX.

```
type
  TRVDocXSaveArea = // defined in RVStyle.pas
   (rv_docxs_TextStyle, rv_docxs_ParaStyle,
    rv_docxs_ListStyle, rv_docxs_StyleTemplate,
    rv_docxs_StyleTemplateText, rv_docxs_StyleTemplatePara,
    rv_docxs_CellProps, rv_docxs_RowProps, rv_docxs_TableProps,
    rv_docxs_SectProps, rv_docxs_Settings, rv_docxs_CoreProperties);
 TRVSaveDocXExtraEvent = procedure (Sender: TCustomRichView;
   Area: TRVDocXSaveArea; Obj: TObject;
   Index1, Index2: Integer; var OOXMLCode: TRVAnsiString [623]) of object;

property OnSaveDocXExtra: TRVSaveDocXExtraEvent;
```

This event occurs when saving DocX (SaveDocX [215], SaveDocXToStream [216]).

Text assigned to **OOXMLCode** parameter will be saved in DocX. This text must be encoded as UTF-8.

Area defines a place where **OOXMLCode** will be inserted. Meaning of **Index1**, **Index2** and **Obj** depends on **Area**.

Area	Meaning	Parent XML Element	XML file
rv_docxs_TextStyle	Occurs when saving a text style. **Index1** is an index of the text style in TextStyles [404] collection, **Obj** is a text style object [444]	<w:rPr>	document files*
rv_docxs_ParaStyle	Occurs when saving a paragraph style. **Index1** is an index of the paragraph style in ParaStyles [398] collection, **Obj** is a paragraph style object [448]	<w:pPr>	document files*

rv_rtfs_ListStyle	Occurs when saving a list style and list levels.
	Index1 is an index of the list style in ListStyles 396.
	Index2 equals to -1 when saving the list description, or it equals to the level index when saving a list level.
	Obj is a list style object 459
		if **Index2** = -1:
		<w:abstractNum>;
		if **Index2** >= 0:
		<w:lvl>
			word/numbering.xml

rv_docxs_StyleTemplate	Occurs when saving a style template as a style sheet item. **Index1** is an index of the style template in StyleTemplates 402. **Obj** is a style template 460.	<w:style>	word/styles.xml
rv_docxs_StyleTemplateText	Occurs when saving text attributes of a style template inside a style sheet item. **Index1** is an index of the style template in StyleTemplates 402. **Obj** is a style template 460.	<w:rPr> inside <w:style>	word/styles.xml
rv_docxs_StyleTemplatePara	Occurs when saving paragraph attributes of a style template inside a style sheet item. **Index1** is an index of the style template in StyleTemplates 402. **Obj** is a style template 460.	<w:pPr> inside <w:style>	word/styles.xml
rv_docxs_CellProps	Occurs when saving table cell properties **Index1** = row index, **Index2** =column index, **Obj** is the table 528.	<w:tcPr>	document files*
rv_docxs_RowProps	Occurs when saving table row properties. **Index1** = row index, **Obj** is the table 528.	<w:trPr>	document files*
rv_docxs_TableProps	Occurs when saving table properties. **Obj** is the table 528.	<w:tblPr>	document files*
rv_docxs_SectProps	Occurs when saving page properties.	<w:sectPr>	word/document.xml
rv_docxs_Settings	Occurs when saving document settings.	<w:settings>	word/settings.xml
rv_docxs_CoreProperties	Occurs when saving document properties.	<cp:coreProperties>	docProps/core.xml

* - "document files" include:

- word/document.xml
- word/footnotes.xml
- word/endnotes.xml
- XML files used for saving headers and footers (header.xml. footer.xml, etc.)

See also properties:

- DocParameters 140.

See also events:

- OnSaveRTFExtra 263.
- OnSaveHTMLExtra 259.

1.4.1.1.4.24 TRichView.OnSaveHTMLExtra

Allows saving additional information in HTML.

```
type
  TRVHTMLSaveArea = (rv_thms_Head, rv_thms_BodyAttribute,
    rv_thms_Body, rv_thms_End);
  TRVSaveHTMLExtraEvent = procedure (
    Sender: TCustomRichView; Area: TRVHTMLSaveArea;
    CSSVersion: Boolean; var HTMLCode: String) of object;

property OnSaveHTMLExtra: TRVSaveHTMLExtraEvent;
```

This event occurs when saving HTML by methods SaveHTML [216] (with **CSSVersion** parameter = False) and SaveHTMLEx [217] (with **CSSVersion**=True).

Text assigned to **HTMLCode** parameter will be saved in HTML. This is a Unicode string for Delphi/C++Builder 2009 or newer, and ANSI string for the older versions of Delphi/C++Builder.

Area can be:

Area	HTMLCode is inserted...
rv_thms_Head	between <head> and </head>
rv_thms_BodyAttribute	between <body and >
rv_thms_Body	just after <body>
rv_thms_End	just before </body>

Example

```
procedure TMyForm.MyRichViewSaveHTMLExtra(
  Sender: TCustomRichView; Area: TRVHTMLSaveArea;
  CSSVersion: Boolean; var HTMLCode: String);
begin
  case Area of
  rv_thms_Head:
    HTMLCode := '<script></script>';
  rv_thms_BodyAttribute:
    HTMLCode := 'alink=#ff0000';
  rv_thms_Body:
    HTMLCode := 'This document is generated by '+
    '<A href="http://www.trichview.com">RichView</A><BR>';
  end;
end;
```

Note: TRVHTMLSaveArea is defined in RVStyle.pas.

See also events:

- OnHTMLSaveImage [241];
- OnSaveImage2 [260];
- OnSaveParaToHTML [263];
- OnSaveRTFExtra [263];
- OnSaveDocXExtra [257].

See also:

- Export to HTML [62].

1.4.1.1.4.25 TRichView .OnSaveImage2

Occurs when TRichView saves image to HTML file or stream.

```
type
  TRVSaveImageEvent2 = procedure (Sender: TCustomRichView;
    Graphic: TGraphic; SaveFormat: TRVSaveFormat;
    const Path, ImagePrefix: String;
    var ImageSaveNo: Integer; var Location: String;
    var DoDefault: Boolean) of object;

property OnSaveImage2: TRVSaveImageEvent2;
```

(introduced in version 1.8; "hidden" parameters are added in version 14)

This event is alternative to OnHTMLSaveImage. It is more easy to use, because the image is available in the Graphic parameter (in OnHTMLSaveImage, you need to extract image from the item or RVData yourself). Besides, this event has **ImagePrefix** and **ImageSaveNo** parameters, so you do not need to store them in global variables.

OnSaveImage2 is not called if:

- *rvsoUseItemImageFileNames* is included in the Options parameter of SaveHTML/SaveHTMLEx and value of *rvespImageFileName* item property is not empty;
- DoDefault parameter of OnHTMLSaveImage was set to *False* for this image.

Input parameters:

Graphic – image to save.

SaveFormat is equal to *rvsfHTML*.

Path – destination directory of HTML file (or **Path** parameter of SaveHTMLToStream/SaveHTMLToStreamEx)

ImagePrefix – recommended beginning of image file name (this is **ImagesPrefix** parameter of SaveHTMLToStream/SaveHTMLToStreamEx)

ImageSaveNo – value that you can use to create unique file names (see GetNextFileName method in the description of OnHTMLSaveImage).

"Hidden" input parameters:

Sender.Style.RVData (must be typecasted to TCustomRVData) – document containing the item that is being saved to an image; it can be **Sender**.RVData, table cell, or RVData of cell inplace-editor.

Sender.Style.ItemNo – index of this item inside **Sender.Style.RVData**. If this event is called for background image (document or cell), **ItemNo**=-1.

Output parameters:

Location – set this parameter to file name of image where you saved it. This string will be inserted in HTML file.

DoDefault – set to *False* if you saved this image yourself. Otherwise, the default image saving procedure will be called.

Note: when saving images from image-lists (*bullets*, *hotspots*, list markers with images), TRichView creates a temporal TBitmap and passes it to this event.

See also events:

- OnHTMLSaveImage;
- OnSaveItemToFile.

See also:

- Export to HTML.

1.4.1.1.4.26 TRichView.OnSaveItemToFile

Allows you to change how document items are saved in text, RTF, DocX or HTML file or stream.

```
type
  TRVSaveItemToFileEvent = procedure (Sender: TCustomRichView;
    const Path: String; RVData: TCustomRVData[608]; ItemNo: Integer;
    SaveFormat: TRVSaveFormat[639]; Unicode: Boolean; var OutStr:TRVRawByteString[638];
    var DoDefault: Boolean) of object;
```

(introduced in v1.8)

This is a "low level" event allowing to change saving of items completely.

There are events offering a subset of functionality of this event:

- OnHTMLSaveImage[241],
- OnSaveImage2[260],
- OnSaveComponentToFile[256].

These events are easier to use than this event.

Input parameters:

Path – path to output file;

RVData, **ItemNo** – item to save. **RVData** – document containing item that is being saved; it can be **Sender**.RVData[151], table cell[567], or RVData of cell inplace-editor. **ItemNo** – index of this item inside **RVData**.

SaveFormat identifies file format, one of *rvsfText*, *rvsfHTML*, *rvsfRTF, rvsfDocX*.

Unicode – *True* when saving Unicode[64] text files, or UTF-8 HTML files, or DocX.

OutStr:

- for non-text items: empty string;
- for text items: text that needs to be saved (it can be a part of the item's text when saving selected fragment). For Unicode text items, this is a "raw Unicode" string..

DoDefault – *True*.

Output parameters:

Set values of **DoDefault** and **OutStr** according to the table below. Mismatched values will cause corrupted files.

Unicode parameter	Output value of DoDefault	For ANSI text items	For Unicode text items	For non-text items
Any	*True*	For text files: default saving, value of **OutStr** is ignored. For HTML, RTF, DocX: value of **OutStr** must be set to ANSI string. This string will be processed afterwards (for example, '<' will be changed to '<' for HTML).	For text files: default saving, value of **OutStr** is ignored. For HTML, RTF, DocX: value of **OutStr** must be set to "raw Unicode" string. This string will be processed afterwards (for example, '<' will be changed to '<' for HTML).	Default saving, value of **OutStr** is ignored.
False	*False*	**OutStr** must be set to ANSI string, this string will be inserted in the file as it is (no control codes will be replaced).		
True	*False*	**OutStr** must be set to "raw Unicode" string, this string will be inserted in the file without further processing (no control codes will be replaced).		

If you do not want to change saving of some item, leave **DoDefault** to *True*.

You cannot override some special processing for paragraph markers when saving to HTML and RTF.

Notes about DocX:

- this event is not called for tables
- this event allows changing saving of numbering sequences[102] only partially

Example 1: saving text items in UTF-8 encoding in HTML and text files.

In this example, we change default saving only if **Unicode** parameter is *False* (it is equal to *False* when saving ANSI text or HTML files).

```
uses RVUni, CRVData;

procedure TMyForm.MyRichViewSaveItemToFile(
  Sender: TCustomRichView; const Path: String;
  RVData: TCustomRVData; ItemNo: Integer;
  SaveFormat: TRVSaveFormat; Unicode: Boolean;
  var OutStr: TRVRawByteString[638]; var DoDefault: Boolean);
begin
  if (SaveFormat in [rvsfText, rvsfHTML[639]]) and
     (RVData.GetItemStyle[194](ItemNo)>=0) and
     not Unicode then
  begin
    if Sender.Style[156].TextStyles[404][RVData.GetItemStyle(ItemNo)].Unicode[446] then
      OutStr := UTF8Encode(RVU_RawUnicodeToWideString(OutStr))
    else
      OutStr := AnsiToUtf8(OutStr);
    DoDefault := False;
  end;
end;
```

Note 1: For HTML files, there is an option for saving as UTF-8, so this code is given only as an example. This example is imperfect, because:

- this code does not include replacement of <, >, &, multiple spaces when saving to HTML; we cannot set **DoDefault** to *True* for HTML if the source item is Unicode, because the default RichView procedure expects "raw Unicode" in this case, but we output UTF-8;
- non-text items are not converted to UTF-8 (so the file will be corrupted if their text representation contains non-English characters);
- HTML encoding is not changed to UTF-8 (so the HTML will be correct only if no encoding is saved at all; it happens in RVStyle.TextStyles[0].Charset=DEFAULT_CHARSET).

Note 2: One more simplification: AnsiToUtf8 uses a system default code page for converting ANSI to Unicode (for better results, languages (Charsets) of text items must be taken into account for the conversion).

Example 2: saving pictures as '<PIC>' (in text files, HTML and RTF)

```
procedure TMyForm.MyRichViewSaveItemToFile(
  Sender: TCustomRichView; const Path: String;
  RVData: TCustomRVData; ItemNo: Integer;
  SaveFormat: TRVSaveFormat; Unicode: Boolean;
  var OutStr: TRVRawByteString[638]; var DoDefault: Boolean);
begin
  if (RVData.GetItemStyle[194](ItemNo)=rvsPicture[663]) then
  begin
    if SaveFormat=rvsfHTML then
      OutStr := '&lt;PIC&gt;'
    else
      OutStr := '<PIC>';
    if Unicode then
      OutStr := RVU_GetRawUnicode(OutStr);
```

```
    DoDefault := False;
  end;
end;
```

Note: for non-text items, **OutStr** is saved as it is; so we need to replace '<' with '<' and '>' with '>' for HTML.

1.4.1.1.4.27 TRichView.OnSaveParaToHTML

Allows to add your HTML code when saving paragraphs to HTML.

```
property OnSaveParaToHTML: TRVSaveParaToHTMLEvent

TRVSaveParaToHTMLEvent =
  procedure (Sender: TCustomRichView;
    RVData: TCustomRVData[608]; ItemNo: Integer;
    ParaStart, CSSVersion: Boolean;
    var HTMLCode: String) of object;
```

(introduced in v1.8)

Input parameters:

ParaStart:

- *True*, if the returned text will be inserted in the beginning of paragraph (before , or <li>),
- *False*, if it will be inserted in the end (after , or).

RVData, **ItemNo** define the location. if **ParaStart**=*True*, **ItemNo** is an index of the first paragraph item (in **RVData**). If *False*, it is an index of the first item of the next paragraph (or **RVData**.ItemCount[144] for the last paragraph).

RVData can be **Sender**.RVData[151], or table cell, or cell inplace editor's RVData[151].

Output parameters:

HTMLCode – output string. This is a Unicode string for Delphi/C++Builder 2009 or newer, and ANSI string for the older versions of Delphi/C++Builder.

Example: each paragraph in its own table

```
procedure TMyForm.MyRichViewSaveParaToHTML(Sender: TCustomRichView;
  RVData: TCustomRVData; ItemNo: Integer; ParaStart,
  CSSVersion: Boolean; var HTMLCode: String);
begin
  if ParaStart then
    HTMLCode := '<table border=1 width=100%><tr><td>'
  else
    HTMLCode := '</td></tr></table>'
end;
```

This event is public, so you need to assign its handler in code at run time.

See also:

- OnSaveHTMLExtra[259];
- Export to HTML[62].

1.4.1.1.4.28 TRichView.OnSaveRTFExtra

Allows saving additional information in RTF

```
type
  TRVRTFSaveArea = // defined in RVStyle.pas
    (rv_rtfs_TextStyle, rv_rtfs_ParaStyle, rv_rtfs_ListStyle,
```

```
    rv_rtfs_StyleTemplate, rv_rtfs_StyleTemplateText,
    rv_rtfs_CellProps, rv_rtfs_RowProps, rv_rtfs_Doc);
 TRVSaveRTFExtraEvent = procedure (Sender: TCustomRichView;
   Area: TRVRTFSaveArea; Obj: TObject;
   Index1, Index2: Integer;
   InStyleSheet: Boolean; var RTFCode: TRVAnsiString) of object;

property OnSaveRTFExtra: TRVSaveRTFExtraEvent;
```

This event occurs when saving RTF (SaveRTF, SaveRTFToStream).

Text assigned to **RTFCode** parameter will be saved in RTF.

Area defines a place where **RTFCode** will be inserted. Meaning of **Index1**, **Index2** and **Obj** depends on **Area**.

Area	Meaning
rv_rtfs_TextStyle	Occurs when saving text style. **Index1** is an index of the text style in TextStyles collection, **Obj** is a text style object If **InStyleSheet**=*True*, this style is saved as a part of style sheet (when RTF is saved with *rvrtfSaveStyleSheet* option)
rv_rtfs_ParaStyle	Occurs when saving paragraph style. **Index1** is an index of the paragraph style in ParaStyles collection, **Obj** is a paragraph style object If **InStyleSheet**=*True*, this style is saved as a part of style sheet (when RTF is saved with *rvrtfSaveStyleSheet* option)
rv_rtfs_ListStyle	Occurs when saving list style and list levels. **Index1** is an index of the list style in ListStyles. **Index2** equals to -1 when saving the list description, or it equals to the level index when saving list level. **Obj** is a list style object
rv_rtfs_StyleTemplate	Occurs when saving a style template as a style sheet item. **Index1** is an index of the style template in StyleTemplates. **Index2** = -1 **Obj** is a style template.
rv_rtfs_StyleTemplateText	Occurs when saving a style template as an additional style sheet item (containing only character properties), linked the the main style sheet item. Such additional style sheet items are created for style templates having Kind = *rvstkParaText*. **Index1** is an index of the style template in StyleTemplates. **Index2** = -1 **Obj** is a style template.
rv_rtfs_CellProps	Occurs when saving table cell properties. **Index1** = row index, **Index2** =column index, **Obj** is the table.
rv_rtfs_RowProps	Occurs when saving table row properties.

	Index1 = row index, **Obj** is the table[528].
rv_rtfs_Doc	Allows saving additional information at the beginning of the document

Example

```
procedure TMyForm.MyRichViewSaveRTFExtra(
  Sender: TCustomRichView; Area: TRVRTFSaveArea;
  Obj: TObject; Index1, Index2: Integer;
  InStyleSheet: Boolean; var RTFCode: TRVAnsiString[623]);
begin
  if Area=rv_rtfs_Doc then
    RTFCode := '\paperw11906\paperh16838\margl567\'+
      'margr567\margt567\margb567';
end;
```

This example saves paper size = A4, and margins = 1 cm.

See also properties:

- DocParameters[140].

See also events:

- OnSaveDocXExtra[257];
- OnSaveHTMLExtra[259].

See also examples:

- ***http://www.trichview.com/forums/viewtopic.php?t=64***.

1.4.1.1.4.29 TRichView .OnSelect

Occurs when selection in the document was changed (not contents, but bounds)

```
property OnSelect:TNotifyEvent;
```

Changes in selection can occur when the user changes the selection using the mouse or keyboard, or when the selection is changed with the methods:

- SelectAll[223];
- Deselect[181];
- SetSelectionBounds[235];
- (and sometimes SelectionExists[224]).

See also:

- Selection in RichView[47].

1.4.1.1.4.30 TRichView .OnSpellingCheck

Allows to mark misspelled words.

```
type
  TRVSpellingCheckEvent = procedure (Sender: TCustomRichView;
    const AWord: String; StyleNo: Integer;
    var Misspelled: Boolean) of object;

property OnSpellingCheck: TRVSpellingCheckEvent;
```

(introduced in v1.9)

This event is called for each word in the document.

Input parameters:

AWord is a word to check (Unicode for Delphi/C++Builder 2009 or newer, ANSI for older versions of Delphi/C++Builder).

StyleNo is a style of text item [88] containing this word (index in the collection **Sender**.Style [156].TextStyles [404]).

Output parameter:

Misspelled, must be set to True if this word is misspelled.

Example (using ***Addict3*** spelling checker):

```
procedure TMyForm.MyRichViewEditSpellingCheck(Sender: TCustomRichView;
  const AWord: String; StyleNo: Integer; var Misspelled: Boolean);
begin
  Misspelled := not MyRVAddictSpell3.WordAcceptable(AWord);
end;
```

This event is called in thread context.

See also:

- Live spelling check in RichView [69].

1.4.1.1.4.31 TRichView.OnStyleTemplatesChange

Occurs when the collection Style [156].StyleTemplates [402] was changed.

```
property OnStyleTemplatesChange: TNotifyEvent;
```

(introduced in version 14)

This event may be called only if UseStyleTemplates [158]=*True*.

It is called:

- when loading or inserting RVF and RTF files or streams (and StyleTemplateInsertMode [156] <>*rvstimIgnoreSourceStyleTemplates*);
- (in editor) by ChangeStyleTemplates [304] method;
- by RichViewActions that can modify this collection (TrvActionNew, TrvActionOpen, TrvActionInsertHyperlink, and actions calling ChangeStyleTemplates [304]: TrvActionStyleTemplates, TrvActionAddStyleTemplate).

1.4.1.1.4.32 TRichView.OnURLNeeded

Occurs when RichView wants to save hyperlink to HTML or RTF file.

```
type
  TRVURLNeededEvent =
    procedure(Sender: TCustomRichView; id: Integer;
    var url:String) of object;

property OnURLNeeded:TRVURLNeededEvent
```

This is an obsolete event.

Use OnWriteHyperlink [267]

1.4.1.1.4.33 TRichView.OnWriteHyperlink

Occurs when RichView wants to save hypertext link to HTML or RTF file

```
type
  TRVWriteHyperlink = procedure (Sender: TCustomRichView;
     id: Integer; RVData: TCustomRVData[608]; ItemNo: Integer;
     SaveFormat: TRVSaveFormat[639];
     var Target, Extras: String) of object;

property OnWriteHyperlink: TRVWriteHyperlink;
```

(introduced in v1.8)

By default, items tags[34] are saved as targets of hypertext links in RTF and HTML.

But you can use this event to customize saving.

In order to do it, you need to assign **Target** parameter to the target URL.

Input parameters:

id – identifier of the hypertext link; this event can be called for unformatted documents; in this case, **id**=-1.

RVData – document containing hyperlink (it may be **Sender**.RVData[151], or table cell, or cell inplace editor's RVData[151])

ItemNo – index of the hyperlink-item inside **RVData**.

SaveFormat can be *rvsfHTML* or *rvsfRTF*.

Output parameters:

Target – hyperlink target to save in file or stream. If you return an empty string, this item will not be saved as a hyperlink.

Extras – optional additional information for saving with the link.

Example 1:

This example does the same work as if the event is not assigned: saves the item tag[34] as a target.

```
procedure TMyForm.MyRichViewWriteHyperlink(
  Sender: TCustomRichView; id: Integer;
  RVData: TCustomRVData; ItemNo: Integer;
  SaveFormat: TRVSaveFormat;
  var Target, Extras: String);
begin
  Target := RVData.GetItemTag[195](ItemNo);
end;
```

Example 2:

The same, but in addition to saving the link target, this example saves the hint "Click here!" (both for RTF and HTML) and the attribute "target=_blank" for HTML.

```
procedure TMyForm.MyRichViewWriteHyperlink(
  Sender: TCustomRichView; id: Integer;
  RVData: TCustomRVData; ItemNo: Integer;
  SaveFormat: TRVSaveFormat;
  var Target, Extras: String);
begin
  Target := RVData.GetItemTag[195](ItemNo);
  case SaveFormat of
    rvsfHTML:
      Extras := 'target=_blank title="Click here!"';
```

```
    rvsfRTF:
      Extras := '\o "Click here!"';
  end;
end;
```

Saving hint is given here only as an example. TRichView saves hints stored in the *rvespHint* item's extra string property [630] automatically.

Example 3:

```
procedure TMyForm.MyRichViewWriteHyperlink(
  Sender: TCustomRichView; id: Integer;
  RVData: TCustomRVData; ItemNo: Integer;
  SaveFormat: TRVSaveFormat;
  var Target, Extras: String);
begin
  case id-FirstJumpNo [144] of
    0: Target := 'http://www.borland.com';
    1: Target := 'http://www.inprise.com';
    2: Target := 'http://www.codegear.com';
    3: Target := 'http://www.embarcadero.com';
  end;
end;
```

See also events:

- OnReadHyperlink [248].

See also methods:

- SaveHTML [216];
- SaveHTMLEx [217];
- SaveRTF [219];
- (and methods saving RTF and HTML to streams).

See also:

- Hypertext [35];
- Saving to HTML [62].

1.4.1.1.5 Classes of Properties

1.4.1.1.5.1 TRVRTFReaderProperties

This is a type of TRichView [128].RTFReadProperties [151]. It contains properties for controlling RTF import in TRichView and TRichViewEdit. Do not create objects of this class yourself, use this property instead.

Unit RVRTFProps.

Syntax

```
TRVRTFReaderProperties = class (TPersistent)
```

Properties

The most important properties are:

- ParaStyleMode [272];
- TextStyleMode [274];
- UnicodeMode [274].

The following properties allow to ignore some part of RTF content:

- IgnoreBookmarks [271];
- IgnoreFields [271];
- IgnoreNotes [271];

- IgnorePictures 271;
- IgnoreSequences 272;
- IgnoreTables 272;
- SkipHiddenText 273;
- ReadDocParameters 273.

Reading Hyperlinks

By default, hyperlinks targets are loaded in items tags 34. Use TCustomRichView.OnReadHyperlink 248 to customize loading.

Reading RTF Images

The following types of RTF pictures are supported (if IgnorePictures 271 = *False*):

- bitmaps (DIB and DDB), loaded as TBitmap;
- metafiles (WMF and EMF), loaded as TMetafile; if metafile contains only a bitmap, it can be loaded as TBitmap, see ExtractMetafileBitmaps 271;
- Jpegs, loaded as TJPEGImage;
- PNG, if PNG class is specified (for Delphi 2009+, TPngImage is used by default; you can specify a PNG class using RVGraphicHandler 662.RegisterPngGraphic);
- external images, loaded from files (TCustomRichView.OnImportPicture 242 event occurs).

Hides grid lines (dotted lines in place of hidden borders) for tables imported from RTF.

```
property AutoHideTableGridLines: Boolean;
```

(introduced in version 1.6)

If this property is set to *True*, RichView includes *rvtoHideGridLines* in Options 540 of tables read from RTF.

Note: it's not necessary to set this option for individual tables. You can hide and show grid for all tables using rvoShowGridLines option in TRichView.Options 146.

Default value:

False

Specifies whether to add base path (path to RTF file) to hyperlink targets.

```
property BasePathLinks: Boolean;
```

(introduced in version 10)

Set to *False* if you do not want to add the document path to relative hyperlinks on RTF loading.

For example, when loading "c:\Docs\MyFile.rtf' containing link to "MyFile2.rtf":

- if this property is *False*, TCustomRichView.OnReadHyperlink 248 event occurs with Target='MyFile2.rtf' (or, if the event is not assigned, this value is stored in them item tag 34);
- if this property is *True*, this event occurs with Target='c:\Docs\MyFile2.rtf' (or, if the event is not assigned, this value is stored in them item tag).

In ***RVHTMLImporter***, this property corresponds to *rvhtmloBasePathLinks* in RVHTMLImporter.Options.

Default value:

True

Specifies a character set 434 for Unicode text read from RTF.

```
property CharsetForUnicode: TFontCharset;
```

(introduced in version 11)

This property is used if UseCharsetForUnicode 275=*True* and TextStyleMode 274<>*rvrsUseSpecified*.

This character set is applied to Unicode text (see UnicodeMode[274]) read from RTF.

Using a single specified character set for all text may be useful if you want to avoid words consisting of several text items (having the same font but different character sets) in UnicodeMode[274]=*rvruOnlyUnicode.*

Default value:

DEFAULT_CHARSET

Specifies the mode of import of text highlighting.

```
type
  TRVRTFHighlight = // defined in RVScroll.pas
    (rtfhlIgnore, rtfhlFixedColors, rtfhlColorTable);

property ConvertHighlight: TRVRTFHighlight;
```

(introduced in version 1.7)

RTF editors show text highlighting as a text background color. But this is not a real text background color (which is always imported by RichView), this is not even a RTF text attribute.

 – this is a MS Word toolbar button applying text highlighting.

Mode	Meaning
rtfhlIgnore	Highlighting is ignored.
rtfhlFixedColors	Highlighting is converted to text background using a fixed set of colors (according to the old RTF specification)
rtfhlColorTable	Highlighting is converted to text background using RTF color table.

MS Word (and TRichView) always puts 16 default colors to the beginning of RTF color table, so for such documents, both *rtfhlFixedColors* and *rtfhlColorTable* work absolutely identically. TRichView itself never saves text background color as a highlight, it uses the special RTF keyword for text background.

It's not recommended to change value of this property.

Default value:

rtfhlColorTable

Contains error code for the last RTF reading operation.

```
ErrorCode: TRVRTFErrorCode;
```

```
type // defined in RVRTFErr unit.
  TRVRTFErrorCode = (
    rtf_ec_OK,              // No error
    rtf_ec_StackUnderflow, // Unmatched '}'
    rtf_ec_StackOverflow,  // Too many '{' -- memory exhausted
    rtf_ec_UnmatchedBrace, // RTF ended during an open group.
    rtf_ec_InvalidHex,     // Invalid hex character found in data
    rtf_ec_BadTable,       // RTF table invalid (internal parser error)
    rtf_ec_Assertion,      // Assertion failure (internal parser error)
    rtf_ec_EndOfFile,      // End of file reached while reading RTF
    rtf_ec_FileOpenError,  // Parser cannot open input file
    rtf_ec_Exception,      // Exception (internal parser error)
```

```
    rtf_ec_InvalidPicture, // Invalid picture in RTF
    rtf_ec_Aborted         // (internal parser error)
    );
```

Option for reading metafiles from RTF

```
property ExtractMetafileBitmaps: Boolean;
```

(introduced in version 1.9)

MS Word usually does not save bitmaps in RTF as they are, it wraps them in metafiles.

If this property is set to *False*, such bitmaps will be loaded as TMetafile. TMetafile sometimes scales bitmap images inaccurately.

If this property is set to *True*, TRichView attempts to extract bitmaps embedded in metafiles and to use them instead of metafiles. If successful, they are inserted as TBitmap.

This option does not affect RTF metafiles containing something besides bitmaps, no bitmaps, or multiple bitmaps – they will be loaded as TMetafile.

Default value:

True

Disallows loading bookmarks from RTF.

```
property IgnoreBookmarks: Boolean;
```

(introduced in version 10)

Bookmarks starts are read as *checkpoints* 30. Bookmark ends cannot be read.

Default value:

False

Disallows loading "page number" 109 and "page count" 110 fields from RTF.

```
property IgnoreFields: Boolean;
```

(introduced in version 15)

If *True*, fields are loaded as a plain text.

Default value:

False

Disallows loading footnotes 104, endnotes 103 and references to them 108 from RTF.

```
property IgnoreNotes: Boolean;
```

(introduced in version 10)

If loading of notes is allowed, their properties are also loaded, specifically Style 156.FootNotePageReset 392.

Default value:

False

Disallows loading pictures from RTF.

```
property IgnorePictures: Boolean;
```

Set it to *True* to import RTF without pictures.

TRichView supports RTF pictures in the following formats:

- Windows Bitmaps (DIB and DDB), loaded as TBitmap;
- Metafiles, loaded as TMetafile or as TBitmap (see ExtractMetafileBitmaps 271 property);

- Enhanced Metafiles, loaded as TMetafile,
- JPEG (D3+), loaded as TJpegImage,
- PNG, see RVGraphicHandler[662].RegisterPngGraphic.

Default value:

False

Disallows loading numbering sequences[102] from RTF.

```
property IgnoreSequences: Boolean;
```

(introduced in version 10)

If this property is *True*, RTF numbered sequences are loaded as a plain text.

Default value:

False

Disallow loading tables from RTF.

```
property IgnoreTables: Boolean;
```

(introduced in version 10)

If *True*, contents of tables are loaded as normal text.

Default value:

False

If set to *True*, all line breaks (added with Shift + Enter) in RTF will be treated as paragraph starts.

```
property LineBreaksAsParagraphs: Boolean;
```

(introduced in version 1.7)

Default value:

False

Defines mode of reading of paragraph properties from RTF.

```
property ParaStyleMode: TRVReaderStyleMode[639];
```

This property defines how RTF paragraph attributes will be converted to TRichView.Style[156].ParaStyles[398].

Mode	Meaning
rvrsUseSpecified	All paragraph formatting in RTF will be ignored, all paragraphs will have style specified in ParaStyleNo[273] property.
rvrsUseClosest	RTF formatting will be mapped to the the most similar existing paragraph style[398]
rvrsAddIfNeeded	This mode provides the most closer look to the original RTF. RichView will try to use existing styles as much as possible, but if style with the desired formatting does not exist, RichView will add a new one to the end of the collection of paragraph styles[398] and use it.

List markers (bullets and numbering) are imported only if ParaStyleMode=*rvrsAddIfNeeded*.

If the both TextStyleMode[274]=**ParaStyleMode**=*rvrsAddIfNeeded*, and TRichView.UseStyleTemplates[158]=*True*, and StyleTemplateInsertMode[156]<>*rvstimIgnoreSourceStyleTemplates*, the RTF style sheet will be read in

TRichView.Style[156].StyleTemplates[402]. For RTF loading, the RTF style sheet replaces existing style templates. For RTF insertion, the RTF style sheet is merged into the existing style templates. In this case, RTF paragraph properties are adjusted according to TRichView.StyleTemplateInsertMode[156].

Default value:

- *rvrsUseClosest* for components created at runtime.
- For components placed on form at designtime, initial value of this property is defined in the component editor[134] for TRichView. By default, the component editor applies "allow adding styles dynamically" mode to new TRichView components, and this property is set to *rvrsAddIfNeeded*.

See also:

- ParaStyleNo[273];
- TextStyleMode[274].

Default paragraph style for RTF loading.

```
property ParaStyleNo: Integer
```

This property is used if ParaStyleMode[272]=*rvrsUseSpecified*.

It is an index in the collection of paragraph styles[398].

Default value:

0

See also:

- ParaStyleMode[272];
- TextStyleNo[274].

Set to True to allow reading DocParameters[140] from RTF.

```
property ReadDocParameters: Boolean;
```

(introduced in version 10)

Default value:

False

This property controls how hidden text is loaded from RTF

```
property SkipHiddenText: Boolean;
```

(introduced in version 1.6)

If *True*, hidden content is not added to RichView.

If *False*, hidden content[42] is added.

Default value:

True

Allows storing file name of external images imported with RTF

```
property StoreImagesFileNames: Boolean;
```

(introduced in version 10)

If you set it to *True*, filenames of external RTF images will be stored in *rvespImageFileName*[630] item property of read images, and you will be able to use them when storing HTML.

Default value:

False

Defines a mode for extraction of character formatting from RTF

```
property TextStyleMode: TRVReaderStyleMode[639];
```

This property defines how RTF text attributes will be converted to TRichView.Style[156].TextStyles[404].

Mode	Meaning
rvrsUseSpecified	All character formatting in RTF will be ignored, all text will have style specified in TextStyleNo[274] property.
rvrsUseClosest	RTF formatting will be mapped to the the most similar existing text style[404].
rvrsAddIfNeeded	This mode provides the most closer look to the original RTF. RichView will try to use existing styles as much as possible, but if style with the desired formatting does not exist, RichView will add a new one to the end of the collection of text styles[404] and use it.

If the both **TextStyMode**=ParaStyleMode[272]=*rvrsAddIfNeeded*, and TRichView.UseStyleTemplates[158]=*True*, and StyleTemplateInsertMode[156]<>*rvstimIgnoreSourceStyleTemplates,* the RTF style sheet will be read in TRichView.Style[156].StyleTemplates[402]. For RTF loading, the RTF style sheet replaces existing style templates. For RTF insertion, the RTF style sheet is merged into the existing style templates. In this case, RTF character properties are adjusted according to TRichView.StyleTemplateInsertMode[156].

Default value:

- *rvrsUseClosest* for components created at runtime.
- For components placed on form at designtime, initial value of this property is defined in the component editor[134] for TRichView. By default, the component editor applies "allow adding styles dynamically" mode to new TRichView components, and this property is set to *rvrsAddIfNeeded*.

See also:

- TextStyleNo[274];
- ParaStyleMode[272].

Default text style for RTF loading.

```
property TextStyleNo: Integer
```

This property is used if TextStyleMode[274]=*rvrsUseSpecified*.

It is an index in the collection of text styles[404].

Default value:

0

See also:

- TextStyleMode[274];
- ParaStyleNo[273].

Defines how text is read from RTF.

```
property UnicodeMode: TRVReaderUnicode;

type
  TRVReaderUnicode =
    (rvruMixed, rvruNoUnicode, rvruOnlyUnicode);
```

Mode	Meaning

rvruMixed	RichView will use Unicode text styles 446 when necessary, trying to use ANSI text as much as possible.
rvruNoUnicode	RichView will ignore Unicode in RTF. Usually RTF contains ANSI alternative for Unicode text, and RichView will use it.
rvruOnlyUnicode	All imported text will be in Unicode encoding. For conversion to Unicode, RichView will use code page identifier read from RTF, or, if it was not found there, TRVStyle.DefCodePage 388 property.

Default value:

For Delphi/C++Builder 2009 or newer: *rvruOnlyUnicode*
For older versions: *rvruNoUnicode*

See also properties:

- UseCharsetForUnicode 275;
- CharsetForUnicode 269.

See also:

- Unicode in RichView 64.

Specifies whether CharsetForUnicode 269 property is used as a character set for Unicode text read from RTF.

```
property UseCharsetForUnicode: Boolean;
```

Default value:

False

Specifies whether

```
property UseCharsetForUnicode: Boolean
```

Set this property to *True* to allow using existing hypertext text styles for non-hypertext text from RTF (in TextStyleMode 274 = *rvrsUseClosest* or *rvrsAddIfNeeded*).

Note: it's not recommended to change value of this property.

Default value:

False

See also:

- Hypertext in RichView 35.

Set it to *True* if you want to use the same values for CellHPadding 536 and CellVPadding 537 for loaded tables.

```
property UseSingleCellPadding: Boolean;
```

(introduced in version 10)

It may be useful if you want to maintain compatibility with older versions of your software, because applications compiled with older versions of TRichView cannot read RVF 59 files containing tables with different cell padding).

Default value:

False

Specifies the document to read a page footer in. Call with *nil* parameter to remove this link.

```
procedure SetFooter(RVData: TCustomRVData[608]; FooterType: TRVHFType[634]=rvhftNormal);
```

Parameters:

RVData – document to read a page footer (of the type specified in **HFType**) in;

HFType – footer type (normal/for the first page/for even pages).

Example:

```
RichView1.Clear[175];
RichView2.Clear[175];
RichView1.RTFReadProperties.ReadDocParameters[273] := True;
RichView1.RTFReadProperties.SetFooter(RichView2.RVData[151]);
RichView1.LoadRTF[209]('test.rtf');
RichView1.Format[182]; // contains document
RichView2.Format[182]; // contains footer
RVPrint1.AssignSource[488](RichView1);
if RichView2.ItemCount[144]>0 then
  RVPrint1.SetFooter[493](RichView2.RVData)
else
  RVPrint1.SetFooter[493](nil);
RVPrint1.AssignDocParameters[488](RichView1.DocParameters[140]);
RVPrint1.FormatPages[490](rvdoAll);
RVPrint1.Print[492]('Doc with footer', 1, False);
```

See also:

- SetHeader[276].

See also properties of TRVStyle[385]:

- MainRVStyle[397].

Specifies the document to read a page header in. Call with *nil* parameter to remove this link.

```
procedure SetHeader(RVData: TCustomRVData[608]; HeaderType: TRVHFType[634]=rvhftNormal);
```

Parameters:

RVData – document to read a page header (of the type specified in **HFType**) in;

HFType – header type (normal/for the first page/for even pages).

Example:

```
RichView1.Clear[175];
RichView2.Clear[175];
RichView1.RTFReadProperties.ReadDocParameters[273] := True;
RichView1.RTFReadProperties.SetHeader(RichView2.RVData[151]);
RichView1.LoadRTF[209]('test.rtf');
RichView1.Format[182]; // contains document
RichView2.Format[182]; // contains header
RVPrint1.AssignSource[488](RichView1);
if RichView2.ItemCount[144]>0 then
  RVPrint1.SetHeader[493](RichView2.RVData)
else
  RVPrint1.SetHeader[493](nil);
RVPrint1.AssignDocParameters[488](RichView1.DocParameters[140]);
RVPrint1.FormatPages[490](rvdoAll);
RVPrint1.Print[492]('Doc with header', 1, False);
```

See also:

- SetFooter[276].

See also properties of TRVStyle [385]:

- MainRVStyle [397].

1.4.1.1.5.2 TRVDocParameters

This is a type of TRichView [128].DocParameters [140]. It contains document's and page properties that can be written to or read from RVF [59] (RichView Format) and RTF (Rich Text Format). The components do not use these properties, but they can be assigned to TRVPrint [479] using the method AssignDocParameters [488].

Unit RVDocParams.

Syntax

```
TRVDocParameters = class(TPersistent)
```

(introduced in version 10)

Properties

This class has the following properties:

- Units [280] – measurement units for margins, paper size, header and footer positions;
- PageWidth [279], PageHeight [279] – page size;
- Orientation [279] – page orientation;
- LeftMargin [278], RightMargin [279], TopMargin [280], BottomMargin [277] – margins;
- MirrorMargins [279] allow to exchange left and right margins on even pages;
- HeaderY [278], FooterY [278] – position of header and footer;
- ZoomMode [281] – zooming mode;
- ZoomPercent [281] – zooming percent.
- Author [277], Title [280], Comments [278] – text properties.

Methods

This class has the following methods:

- Reset [281] resets all properties to default values;
- ResetLayout [281] resets all sizes to default values measured in Units [280];
- UnitsPerInch [281] allows to convert between measurement units;
- ConvertToUnits [281] converts all sizes to the specified units.

Author of the document

```
property Author: String;
```

(introduced in version 13)

In TRichViewEdit [282], you can change this property as an editing operation, see TRichViewEdit.SetStrPropertyEd [337].

Default value:

" (empty string)

Bottom margin, measured in Units [280].

```
property BottomMargin: TRVLength [636];
```

This property is not used by TRichView components, but it can be written and read from RVF [59] (RichView Format) and RTF (Rich Text Format) files.

TRVPrint.AssignFromDocParameters [488] assigns this property to TRVPrint.Margins [486].Bottom.

In TRichViewEdit [282], you can change this property as an editing operation, see TRichViewEdit.SetFloatPropertyEd [337].

Default value:

1.0

Comments to the document

```
property Comments: String;
```

(introduced in version 13)

In TRichViewEdit[282], you can change this property as an editing operation, see TRichViewEdit.SetStrPropertyEd[337].

Default value:

" (empty string)

Instructs to use different headers and footers for odd and even pages.

```
property FacingPages: Boolean;
```

This property is not used by TRichView components, but it can be written and read from RVF[59] (RichView Format) and RTF (Rich Text Format) files.

TRVPrint.AssignFromDocParameters[488] assigns this property to TRVPrint.FacingPages[484].

In TRichViewEdit[282], you can change this property as an editing operation, see TRichViewEdit.SetIntPropertyEd[337].

Default value:

False

Distance from the bottom of page to the bottom of footer, measured in Units[280].

```
property FooterY: TRVLength[636];
```

This property is not used by TRichView components, but it can be written and read from RVF[59] (RichView Format) and RTF (Rich Text Format) files.

TRVPrint.AssignFromDocParameters[488] assigns this property to TRVPrint.FooterY[485].

In TRichViewEdit[282], you can change this property as an editing operation, see TRichViewEdit.SetFloatPropertyEd[337].

Default value:

0.5

Distance from the top of page to the top of header, measured in Units[280].

```
property HeaderY: TRVLength[636];
```

This property is not used by TRichView components, but it can be written and read from RVF[59] (RichView Format) and RTF (Rich Text Format) files.

TRVPrint.AssignFromDocParameters[488] assigns this property to TRVPrint.HeaderY[485].

In TRichViewEdit[282], you can change this property as an editing operation, see TRichViewEdit.SetFloatPropertyEd[337].

Default value:

0.5

Left margin, measured in Units[280].

```
property LeftMargin: TRVLength[636];
```

This property is not used by TRichView components, but it can be written and read from RVF[59] (RichView Format) and RTF (Rich Text Format) files.

TRVPrint.AssignFromDocParameters[488] assigns this property to TRVPrint.Margins[486].Left.

In TRichViewEdit[282], you can change this property as an editing operation, see TRichViewEdit.SetFloatPropertyEd[337].

Default value:

1.25

Instructs to swap left and right margins on even pages.

```
property MirrorMargins: Boolean;
```

This property is not used by TRichView components, but it can be written and read from RVF[59] (RichView Format) and RTF (Rich Text Format) files.

TRVPrint.AssignFromDocParameters[488] assigns this property to TRVPrint.MirrorMargins[487].

In TRichViewEdit[282], you can change this property as an editing operation, see TRichViewEdit.SetIntPropertyEd[337].

Default value:

False

Page orientation.

```
property Orientation: TPrinterOrientation;
```

This property is not used by TRichView components, but it can be written and read from RVF[59] (RichView Format) and RTF (Rich Text Format) files.

In TRichViewEdit[282], you can change this property as an editing operation, see TRichViewEdit.SetIntPropertyEd[337].

Default value:

poPortrait

Page height, measured in Units[280].

```
property PageHeight: TRVLength[636];
```

This property is not used by TRichView components, but it can be written and read from RVF[59] (RichView Format) and RTF (Rich Text Format) files.

Default value:

11.692913386 (A4 height in inches)

Page width, measured in Units[280].

```
property PageWidth: TRVLength[636];
```

This property is not used by TRichView components, but it can be written and read from RVF[59] (RichView Format) and RTF (Rich Text Format) files.

Default value:

8.2677165354 (A4 width in inches)

Right margin, measured in Units[280].

```
property RightMargin: TRVLength[636];
```

This property is not used by TRichView components, but it can be written and read from RVF[59] (RichView Format) and RTF (Rich Text Format) files.

TRVPrint.AssignFromDocParameters[488] assigns this property to TRVPrint.Margins[486].Right.

In TRichViewEdit[282], you can change this property as an editing operation, see TRichViewEdit.SetFloatPropertyEd[337].

Default value:

1.25

Title of the document

```
property Title: String;
```

(introduced in version 13)

In TRichViewEdit[282], you can change this property as an editing operation, see TRichViewEdit.SetStrPropertyEd[337].

Default value:

" (empty string)

Instructs to use a special header and footer on the first page.

```
property FacingPages: Boolean;
```

This property is not used by TRichView components, but it can be written and read from RVF[59] (RichView Format) and RTF (Rich Text Format) files.

TRVPrint.AssignFromDocParameters[488] assigns this property to TRVPrint.TitlePage[487].

In TRichViewEdit[282], you can change this property as an editing operation, see TRichViewEdit.SetIntPropertyEd[337].

Default value:

False

Top margin, measured in Units[280].

```
property TopMargin: TRVLength[636];
```

This property is not used by TRichView components, but it can be written and read from RVF[59] (RichView Format) and RTF (Rich Text Format) files.

TRVPrint.AssignFromDocParameters[488] assigns this property to TRVPrint.Margins[486].Top.

In TRichViewEdit[282], you can change this property as an editing operation, see TRichViewEdit.SetFloatPropertyEd[337].

Default value:

1.0

Measurement units for all sizes in TRVDocParameters.

```
property Units: TRVUnits[648];
```

Assignment to this property does not change other properties. Use ConvertToUnits[281] to convert all sizes to new units.

Use UnitsPerInch[281] to convert values between units.

In TRichViewEdit[282], you can change this property as an editing operation, see TRichViewEdit.SetIntPropertyEd[337].

Default value

rvuInches

Scaling mode

```
type // defined in RVStyle unit
  TRVZoomMode = (rvzmFullPage, rvzmPageWidth, rvzmCustom);
property ZoomMode : TRVZoomMode;
```

This property is not used by TRichView components, but it can be written and read from RVF [59] (RichView Format) and RTF (Rich Text Format) files.

Default value:

rvzmCustom

View scale, % (if ZoomMode [281]=*rvzmCustom*).

```
property ZoomPercent: Integer;
```

This property is not used by TRichView components, but it can be written and read from RVF [59] (RichView Format) and RTF (Rich Text Format) files.

Default value:

100

Converts all sizes to **AUnits**.

```
procedure ConvertToUnits(AUnits: TRVUnits [280]);
```

This method converts all size properties from Units [280] to **AUnits**, then assigns Units [280] = **AUnits**.

The following properties are converted:

- PageWidth [279], PageHeight [279] – page size;
- LeftMargin [278], RightMargin [279], TopMargin [280], BottomMargin [277] – margins;
- HeaderY [278], FooterY [278] – position of header and footer.

Resets all properties to default values.

```
procedure Reset;
```

Resets all sizes to default values measured in Units [280].

```
procedure ResetLayout;
```

It makes sense to call this method on "File | New" command.

The following properties are reset:

- PageWidth [279], PageHeight [279] – page size;
- LeftMargin [278], RightMargin [279], TopMargin [280], BottomMargin [277] – margins;
- HeaderY [278], FooterY [278] – position of header and footer.

In addition, this method resets Orientation [279] to *poPortrait* and MirrorMargins [279] to *False*.

Returns the number of **Units** in one inch.

```
function UnitsPerInch(Units: TRVUnits [280]): TRVLength [636];
```

This method can be used to convert between measurement units.

For example, this code converts from points to millimeters:

```
ValueMM := ValuePoints * UnitsPerInch(rvuMillimeters) / UnitsPerInch(rvuPoints);
```

See the note about *rvuPixels* in the topic about Units [280].

1.4.1.1.6 Examples

1.4.1.1.6.1 Loading UTF-8 files

This example shows how to load Unicode UTF-8 file:

```
procedure LoadUTF8(rv: TCustomRichView[128]; const FileName: String;
  StyleNo, ParaNo: Integer);
var Stream: TFileStream;
    s: TRVRawByteString[638];
    ws: TRVUnicodeString[647];
begin
  Stream := TFileStream.Create(FileName, fmOpenRead);
  SetLength(s, Stream.Size);
  Stream.ReadBuffer(PRVAnsiChar(s)^, Stream.Size);
  Stream.Free;
  rv.Clear;
  ws := UTF8Decode(s);
  rv.AddTextNLW[171](ws, StyleNo, ParaNo, ParaNo, False);
end;
```

Call:

```
LoadUTF8(RichViewEdit1, 'test.txt', 0, 0);
RichViewEdit1.Format[182];
```

If the StyleNo-th style has Unicode[446] property = *True*, the file will be loaded as it is. If not, it will be converted to ANSI text.

See also:

- Unicode in TRichView[64].

1.4.1.2 TRichViewEdit

TRichViewEdit is a control for editing documents with pictures, tables, Delphi controls and hyperlinks.

Unit RVEdit

Syntax

```
TCustomRichViewEdit = class(TCustomRichView[128])
TRichViewEdit = class(TCustomRichViewEdit)
```

RichViewEdit does not display or edit its content if it is not linked to RVStyle component via Style[156] property.

The most important property settings can be done in the component editor[134] for TRichView. In Delphi (C++Builder) IDE, right click the RichViewEdit object on the form, choose "Settings" in the context menu.

Some abilities of TRichView are disabled in TRichViewEdit, such as:

- OnCheckpointVisible[238] event;
- all features related to hypertext ids:OnRVMouseMove[254], OnJump[247], GetJumpPointItemNo[196] work only when user presses and holds Ctrl key, if *rvtoCtrlJumps* in EditorOptions[291] (or if ReadOnly[294]=*True*).

Hierarchy

TObject
TPersistent
TComponent
TControl
TWinControl
TCustomControl
TRVScroller[508]
TCustomRichView[128]
TCustomRichViewEdit

List of properties, events and methods

grouped in several categories (some entries can be in several groups)

- Properties 283
- Events 283
- General methods 283
- Inserting items in the position of caret 283
- Clipboard operations 284
- Advanced methods for modifying items 284
- Information about item at the position of caret 284
- Modifying item at the position of caret 285
- Style templates 285
- Undo/Redo 285
- Unicode 286
- Live spelling check 286
- "Smart Popups" 286
- Related to tables and special items 286
- Working with *checkpoints* 286

Properties

- CurItemNo 288 – index of item in the position of caret, OffsetInCurItem 294 – offset of caret in this item;
- CurItemStyle 289 – style (or type) of the item at the position of caret;
- CurParaStyleNo 289 – current paragraph style index;
- CurTextStyleNo 290 – current text style index, see also ActualCurTextStyleNo 288;
- EditorOptions 291 – options for editing;
- ReadOnly 294 disables/enables editing;
- Modified 293 – all editing methods set it to *True*.
- CustomCaretInterval 290 – a blinking interval for a custom caret.
- AcceptDragDropFormats, AcceptPasteFormats 287 – lists of formats that the editor can accept.
- DefaultPictureVAlign 291 – alignment for pasted and dropped pictures.
- ForceFieldHighlight 293 affects highlighting of label items 101.

See also properties for undo/redo.

Events

- OnChange 357 occurs when the document is changed;
- OnCheckStickingItems 357 allows to add additional text protection;
- OnCurParaStyleChanged 358 occurs when the index of the current paragraph style is changed;
- OnCurTextStyleChanged 358 occurs when the index of the current text style is changed;
- OnPaste 365 allows pasting data from the Clipboard in custom formats;
- OnStyleConversion 366 occurs while executing ApplyStyleConversion 299;
- OnParaStyleConversion 364 occurs while executing ApplyParaStyleConversion 298;
- OnCaretGetOut 356 occurs when the user moves the caret "outside" the editor;
- OnCaretMove 356 occurs when the caret is moved;
- OnItemTextEdit 360 occurs when a text item 88's text is changed as a result of editing operation;
- OnSmartPopupClick 366 occurs when user clicks a "smart popup" 286 button.
- OnMeasureCustomCaret 360 and OnDrawCustomCaret 358 allow drawing your own caret instead of the system one.

General Methods

- ApplyParaStyle 297, ApplyTextStyle 300 apply paragraph and text style to the selection;
- ApplyStyleConversion 299 applies custom style conversion to the selected text;
- ApplyParaStyleConversion 298 applies custom paragraph style conversion to the selected paragraphs;
- Change 303 generates OnChange 357 event;
- BeginUpdate 302, EndUpdate 307 disable and enable redrawing;
- SearchText, -A, -W 336 search for substring.

Inserting Items in the Position of Caret (Insert*** methods)

- InsertText, -A, -W 327, InsertStringTag, InsertStringATag, InsertStringWTag 326 insert text;
- InsertBreak 317 inserts *break* (horizontal line);
- InsertBullet 317 inserts *bullet*;

- InsertControl[318] inserts Delphi control;
- InsertHotspot[319] inserts *hotspot*;
- InsertPicture[322] inserts picture;
- InsertHotPicture[319] inserts picture-hypertext link;
- InsertTab[326] inserts tabulator;
- InsertRVFFromFileEd[324], InsertRVFFromStreamEd[325] insert RVF (RichView Format) from file or stream;
- InsertRTFFromFileEd[323], InsertRTFFromStreamEd[323] insert RTF (Rich Text Format) from file or stream;
- InsertTextFromFile[328],InsertOEMTextFromFile[321],InsertTextFromFileW[328] insert text from file;
- InsertItem[320] – general method for inserting items, inserts one item (usually used for tables).

Clipboard Operations

Main methods and events:

- CanPaste[303] – "can something be pasted from the Clipboard?";
- CanPasteRVF[303] – "can RVF (RichView Format) be pasted from the Clipboard?";
- CanPasteRTF[303] – "can RTF (Rich Text Format) be pasted from the Clipboard?";
- CutDef[306] cuts selection to the Clipboard;
- Paste[328] pastes from the Clipboard;
- PasteBitmap[329] pastes bitmap from the Clipboard;
- PasteMetafile[330] pastes metafile from the Clipboard;
- PasteGraphicFile[329] pastes metafile from the Clipboard;
- PasteRVF[331] pastes RVF (RichView Format) from the Clipboard;
- PasteRTF[330] pastes RTF (Rich Text Format) from the Clipboard;
- PasteTextA[331] pastes ANSI text from the Clipboard;
- PasteTextW[331] pastes Unicode text from the Clipboard;
- event OnPaste[365] allows to paste data from the Clipboard in custom formats.

Also:

- DeleteSelection[306] deletes selection;
- SelectCurrentWord[337] selects word at the position of caret;
- SelectCurrentLine[337] selects the line containing caret.

Advanced Methods for Modifying Items (Set***InfoEd methods)

Main methods:

- SetBreakInfoEd[339] – for changing *break*;
- SetBulletInfoEd[339] – for changing *bullet*;
- SetControlInfoEd[341] – for changing inserted control;
- SetHotspotInfoEd[349] – for changing *hotspot*;
- SetPictureInfoEd[354] – for changing picture or *hot picture*.

Also:

- SetItemExtraIntProperty[Ex]Ed[350], SetItemExtraStrProperty[Ex]Ed[351] – for changing additional properties of items;
- SetItemTagEd[352] – for changing *tag*;
- SetItemVAlignEd[353] – for changing item position relative to the line;
- SetItemTextEd[352] changes text (name) of item;
- SetCheckpointInfoEd[340] – for changing *checkpoint*;
- RemoveCheckpointEd[333] – for deleting *checkpoint*;
- ResizeControl[335] – for resizing inserted control;
- AdjustControlPlacement[296] returns control to the proper position;
- AdjustControlPlacement2[296] returns control to the proper position.

Information about the Item at the Position of Caret (GetCurrent***Info methods)

These methods are equivalent to Get***Info(CurItemNo[288],...) but work also in inplace editors of tables[111].

Main methods:

- GetCurrentTextInfo[316] – about text item;

- GetCurrentBreakInfo 308 – about *break*;
- GetCurrentBulletInfo 308 – about *bullet*;
- GetCurrentControlInfo 309 – about inserted control;
- GetCurrentHotspotInfo 310 – about *hotspot*;
- GetCurrentPictureInfo 315 – about picture or *hot picture*;
- GetCurrentItem 311 – as an object about any item (usually for tables 111);
- GetCurrentItemEx 311.

Also:

- GetCurrentItemExtraIntProperty[Ex] 312, GetCurrentItemExtraStrProperty[Ex] 313 return value of additional item property;
- GetCurrentTag 316 returns item's *tag*;
- GetCurrentItemVAlign 314 returns item position relative to the line;
- GetCurrentCheckpoint 309 returns item's *checkpoint*;
- GetCurrentItemText 313 returns text/name of item;
- GetCurrentLineCol 314 returns line and column at the caret position.

Modifying Item in Position of Caret (SetCurrent***Info methods)

These methods are equivalent to Set***InfoEd(CurItemNo 288,...) but work also in inplace editors of tables 111.

Main methods:

- SetCurrentBreakInfo 342 modifies *break*;
- SetCurrentBulletInfo 342 modifies *bullet*;
- SetCurrentHotspotInfo 345 modifies *hotspot*;
- SetCurrentControlInfo 344 modifies inserted control;
- SetCurrentPictureInfo 348 modifies picture or *hot picture*.

Also:

- SetCurrentItemExtraIntProperty[Ex] 345,
- SetCurrentItemExtraStrProperty[Ex] 351 change value of additional item property;
- SetCurrentTag 349 modifies item's *tag*;
- SetCurrentItemVAlign 348 changes item position relative to the line;
- SetCurrentItemText 347 changes text (name) of item;
- SetCurrentCheckpointInfo 343 modifies *checkpoint*;
- RemoveCurrentCheckpoint 334 deletes *checkpoint*;
- ResizeCurrentControl 335 returns control to the proper position in document;

- RemoveCurrentPageBreak 334 removes "explicit page break before item" flag;
- InsertPageBreak 321 sets "explicit page break before item" flag; can split current item into two parts.

Style templates

Methods:

- ApplyStyleTemplate 299, ApplyTextStyleTemplate 301, ApplyParaStyleTemplate 298 apply the specified style template to the selection;
- ChangeStyleTemplates 304 allows changing the collection of style templates as an editing operation.

Undo/Redo

Main:

- property UndoLimit 295 sets capacity of undo buffer;
- UndoAction 355 returns which operation will be undone next;
- Undo 355 undoes the last operation;
- UndoName 355 returns name of custom undo operation which will be undone next;
- RedoAction 333 returns which operation will be redone next;
- Redo 332 redoes the last undone operation;
- RedoName 333 returns name of custom redo operation which will be redone next;
- ClearUndo 305 clears undo and redo buffers.

Also:

- BeginUndoGroup 302,
- BeginUndoCustomGroup 302,
- SetUndoGroupMode 355.

Unicode

- PasteTextW 331 pastes Unicode text from the Clipboard;
- InsertTextW 327 inserts Unicode text;
- InsertTextFromFileW 328 inserts Unicode text from file.

Live Spelling Check

- GetCurrentMisspelling 315 returns misspelled word at the caret position;
- property LiveSpellingMode 293 controls when live spelling thread starts.

"Smart Popups"

"Smart popups" are buttons that can be used to set properties for the current (at the position of caret) item.

Properties:

- SmartPopupProperties 295 – popup button properties;
- SmartPopupVisible 295 – popup button properties.

Events:

- OnSmartPopupClick 366 occurs when user clicks "smart popup" button.

Related to Tables 528 and Special Items

- InsertItem 320 – general method for inserting items, inserts one item (usually used for tables 98, labels 101, sequences 102, footnotes 104, endnotes 103, references to notes 108).
- GetCurrentItem 311 returns item at the position of caret as an object (usually used for tables 98, labels 101, sequences 102, footnotes 104, endnotes 103, references to notes 108);
- BeginItemModify 301, EndItemModify 307 are used to reformat editor after operations on item (usually on tables);
- CanChange 302 checks possibility of performing editing operation (usually it is called before making operations on table).

Working with *Checkpoints* 30

Editing-style analogs of methods of TRichView 128:

- SetCheckpointInfoEd 340 – adds *checkpoint* for the specified item;
- RemoveCheckpointEd 333 – deletes *checkpoint* for the specified item.

Methods working with the item at the position of caret:

- GetCurrentCheckpoint 309 – returns *checkpoint* associated with the item at the caret position;
- SetCurrentCheckpointInfo 343 – adds/modifies *checkpoint* for the item at the caret position;
- RemoveCurrentCheckpoint 334 – deletes *checkpoint* for the item at the caret position.

Methods working with the checkpoint exactly at the position of caret:

- InsertCheckpoint 318 – inserts *checkpoint* in the caret position;
- GetCheckpointAtCaret 307 – returns the *checkpoint* exactly at the caret position;
- RemoveCheckpointAtCaret 333 – deletes the *checkpoint* exactly at the caret position.

1.4.1.2.1 Properties

1.4.1.2.1.1 TRichView Edit.AcceptDragDropFormats, AcceptPasteFormats

AcceptDragDropFormats lists formats that can be accepted when dropping data in TRichViewEdit.

AcceptPasteFormats lists formats that can be accepted when pasting in TRichViewEdit.

```
property AcceptDragDropFormats: TRVDragDropFormats;
property AcceptPasteFormats: TRVDragDropFormats;

type
  TRVDragDropFormat = (
    rvddRVF, rvddRTF, rvddText, rvddUnicodeText,
    rvddBitmap, rvddMetafile, rvddURL, rvddFiles, rvddHTML);
  TRVDragDropFormats = set of TRVDragDropFormat;
```

(introduced in v1.8 & v14)

Value	D&D Format	Meaning
rvddRVF	'RichView Format'	RichView Format 59
rvddRTF	'Rich Text Format'	RTF
rvddText	CF_TEXT	Plain ANSI text
rvddUnicodeText	CF_UNICODETEXT	Plain Unicode text
rvddBitmap	CF_DIB or CF_BITMAP	Bitmap
rvddMetafile	CF_ENHMETAFILE	Metafile
rvddURL	'UniformResourceLocator'	URL. If available, URL caption is read from CFSTR_FILEDESCRIPTOR format (otherwise caption = target). OnReadHyperlink 248 event occurs with DocFormat=*rvlfURL*. If this event is not processed, URL is inserted as a plain text.
rvddFiles	CF_HDROP	By default, RichViewEdit inserts: ▪ graphic files, ▪ *.RTF (only drag&drop), ▪ *.RVF (only drag&drop), ▪ *.TXT (only drag&drop). You can modify the default processing in OnDropFiles 359 event (only drag&drop).
rvddHTML		Reserved for future use

Example (processing OnReadHyperlink 248 with *rvddURL* support)

```
// OnReadHyperlink
procedure TMyForm.MyRichViewEditReadHyperlink(Sender: TCustomRichView;
  const Target, Extras: String; DocFormat: TRVLoadFormat;
  var StyleNo: Integer; var ItemTag: TRVTag 645; var ItemName: TRVRawByteString 638);
begin
  if DocFormat=rvlfURL then
    StyleNo := GetHyperlinkStyleNo(Sender);
  ItemTag := Target;
end;
```

```
// This function returns the index of a text style having all properties
// like the current style, but hypertext, blue, underlined.
// If this style does not exist, it is added.
function TMyForm.GetHyperlinkStyleNo(rve: TCustomRichViewEdit): Integer;
var TextStyle: TFontInfo[444];
begin
  Result := rve.CurTextStyleNo[290];
  TextStyle := TFontInfo.Create(nil);
  try
    TextStyle.Assign[443](rve.Style[156].TextStyles[404][Result]);
    TextStyle.Jump[445] := True;
    TextStyle.Color[435] := clBlue;
    TextStyle.Style[441] := TextStyle.Style+[fsUnderline];
    Result := rve.Style.FindTextStyle[407](TextStyle);
  finally
    TextStyle.Free;
  end;
end;
```

Default value:

[*rvddRVF, rvddRTF, rvddText, rvddUnicodeText, rvddBitmap, rvddMetafile, rvddFiles*] (all except for *rvddURL*)

See also:

- Paste[328];
- CanPaste[303];
- Drag&Drop[55].

1.4.1.2.1.2 TRichView Edit.ActualCurTextStyleNo

Index of the current text style in the collection of text styles. Can return rvsDefStyle[663].

```
property ActualCurTextStyleNo: Integer;
```

(introduced in v1.8)

This is an index in the collection Style[156].TextStyles[404].

This property is the same as CurTextStyleNo[290], but:

- read-only;
- does not process *rvsDefStyle*[663], returns it as it is.

1.4.1.2.1.3 TRichView Edit.CurItemNo

Index of item at the position of caret

```
property CurItemNo: Integer;
```

This value is in range from 0 to ItemCount[144]-1.

If the caret is inside table cell, this property returns index of the table in the main document. Items of subdocuments (table cells[567]) are not included in the items range of the main document. If you want to know index of item at the position of caret in the table cell, use TopLevelEditor[295].CurItemNo.

This property must be accessed only when the document is formatted.

When called for an empty editor (i.e. ItemCount[144]=0), the method returns -1.

If you want to get the caret position as a number of characters from the beginning of the document, use RVGetCaretPos from RVLinear[621] unit instead.

See also properties:

- OffsetInCurItem[294];

- TopLevelEditor 295;
- CurItemStyle 289;
- CurTextStyleNo 290;
- CurParaStyleNo 289.

See also events:

- OnCaretMove 356.

See also:

- Inserting in the position of the caret 52;
- How to: move the caret to the beginning or to the end of document in TRichViewEdit 668.

1.4.1.2.1.4 TRichView Edit.CurItemStyle

Style (i.e. type) of item at the position of caret

```
property CurItemStyle: Integer;
```

The same as TopLevelEditor 295.GetItemStyle 194(TopLevelEditor 295.CurItemNo 288).

If the current item is a text item, this function returns zero or positive value: index in the collection of text styles (Style 156.TextStyles 404).

Otherwise, this function returns a type of this item: see rvsXXX constants 663.

This property must be accessed only when the document is formatted.

See also methods:

- GetItemStyle 194.

See also properties:

- CurItemNo 288;
- CurTextStyleNo 290.

See also:

- Item types 29.

1.4.1.2.1.5 TRichView Edit.CurParaStyleNo

Index of the current paragraph style in the collection of paragraph styles.

```
property CurParaStyleNo: Integer;
```

This is an index in the collection Style 156.ParaStyles 398.

This is a style of paragraph containing caret.

Value of this is changed when the user moves the caret to another paragraph or edits the document, or when some methods are called. When this property is changed, OnCurParaStyleChanged 358 event occurs.

Usually you do not need not to assign value to this property directly, use ApplyParaStyle 297 or ApplyParaStyleConversion 298 methods instead.

This property must be accessed only when the document is formatted.

See also methods:

- ApplyParaStyle 297;
- ApplyParaStyleConversion 298.

See also properties:

- CurTextStyleNo 290.

See also events:

- OnCurParaStyleChanged 358.

See also:

- Working with paragraphs 37.

1.4.1.2.1.6 TRichView Edit.CurTextStyleNo

Index of the current text style in the collection of styles.

```
property CurTextStyleNo: Integer;
```

This is an index in the collection Style[156].TextStyles[404].

This text style will be used when inserting new text in the document in the position of caret (both by user keyboard input and by methods, such as PasteText[331] or InsertText[327]).

This property is changed when user moves the caret to another text item or edits the document, or when some methods are called. When this property is changed, OnCurTextStyleChanged[358] event occurs.

Value of this property is always positive. To get style (i.e. type) of item at the caret position, use CurItemStyle[289] property.

The editor never sets this property to index of text style having *rvprDoNotAutoSwitch* in Protection[439]. You can do it yourself to insert a special symbol (by keyboard input, or InsertText[327]), but after that the component automatically switches this property back to the last used style without *rvprDoNotAutoSwitch* in Protection[439].

Usually you do not need to assign value to this property directly, use ApplyTextStyle[300] or ApplyStyleConversion[299] instead.

This property must be accessed only when the document is formatted.

This property never returns *rvsDefStyle*[663] constant. It "translates" it and returns the index of text style that is used for text. If you want to know if the text actually has *rvsDefStyle*[663] style, use ActualCurTextStyleNo[288] property.

See also methods:

- ApplyTextStyle[300];
- ApplyStyleConversion[299].

See also properties:

- CurItemStyle[289];
- CurParaStyleNo[289];
- ActualCurTextStyleNo[288].

See also events:

- OnCurTextStyleChanged[358].

1.4.1.2.1.7 TRichView Edit.CustomCaretInterval

If set to positive value, activates the custom caret feature. Defines the timer interval for redrawing caret, in milliseconds.

```
property CustomCaretInterval: Integer;
```

(introduced in version 10)

The caret is activated when you call Format[182] method (and is deactivated when you call Clear[175]).

It's not recommended to use the custom caret feature together with image animation[56].

Default value:

0 (the system caret is used)

See also events:

- OnMeasureCustomCaret[360];
- OnDrawCustomCaret[358].

1.4.1.2.1.8 TRichView Edit.DefaultPictureVAlign

Specifies the default alignment for pasted images, and for images inserted as a result of drag and drop operations.

```
property DefaultPictureVAlign: TRVVAlign[648];
```

(introduced in version 14)

The following methods use this property:

- PasteBitmap[329]
- PasteMetafile[330]
- PasteGraphicFile[329]
- Paste[328] (when inserting a bitmap, a metafile, or a graphic file);

This property is also used when a picture is pasted using the keyboard (when the user presses Ctrl + V or Shift + Insert), or when dropping a picture in the editor.

Default value:

rvvaBaseline

1.4.1.2.1.9 TRichView Edit.EditorOptions

Additional options affecting editing

```
type
  TRVEditorOption = (rvoClearTagOnStyleApp, rvoCtrlJumps,
    rvoDoNotWantReturns, rvoWantTabs,
    rvoAutoSwitchLang, rvoHideReadOnlyCaret,
    rvoDoNotWantShiftReturns, rvoNoImageResize,
    rvoNoCaretHighlightJumps, rvoNoReadOnlyJumps)
  TRVEditorOptions = set of TRVEditorOption;

property EditorOptions: TRVEditorOptions;
```

Options for disabling/enabling certain editing operations

Option	Meaning
rvoDoNotWantReturns	If set, editor ignores Enter keys.
rvoDoNotWantShiftReturns	If set, Shift+Enter does nothing (by default, it inserts a line break inside paragraph – noticeable in paragraphs with bullets or borders)
rvoWantTabs	If set (default), editor accepts Tab key (Ctrl+Tab always works). Tabs can be inserted as several spaces, or as a tabulator[89] (special item type), depending on the value of SpacesInTab[402] property of the linked TRVStyle component.
rvoNoImageResize	Prevents resizing pictures[90] and controls[94] with mouse.

Visual options

Option	Meaning
rvoNoCaretHighlightJumps	If set, hyperlinks will not be highlighted (with HoverColor[436] and HoverBackColor[436]) when the caret is moved inside them.
rvoHideReadOnlyCaret	If set, the editor hides the caret in read-only mode[294]

Other options

Option	Meaning
rvoCtrlJumps	If set (default), the editor switches to the hypertext mode (highlighting hypertext items under mouse cursor, using hypertext cursors, generating OnJump[247] and OnRVMouseMove[254]) when user presses and holds Ctrl key The editor switches back to the editing mode when user releases Ctrl key or changes document, or if the editor loses focus.
rvoNoReadOnlyJumps	If not set (default), the editor automatically switches to the hypertext mode when it becomes read-only. If set, hyperlinks in a read-only mode work like in a normal mode.
rvoClearTagOnStyleApp	If set, editor clears *tags*[34] (set to '') of all text items when changing their style index using ApplyTextStyle[300] or ApplyStyleConversion[299] method.
rvoAutoSwitchLang	If set, the editor processes changing of keyboard locale (WM_INPUTLANGCHANGE message) and switches the current text style[290] to the style having charset[434] of the new keyboard locale. This option helps to provide multilingual input without using Unicode[64]. This option is ignored if the current text style is Unicode[446] style. If not, the editor searches for non-Unicode text styles[404] with Charset[434] of the new keyboard locale, and switches CurTextStyleNo[290] to it. If such style was not found, the editor adds a new style in the collection of text styles. Thus this option may add new styles in the collection of text styles. If you use this option, you need to store text styles with the document[134] (see "allow adding styles dynamically").

Default value (changed in v1.9):

[*rvoWantTabs*, *rvoCtrlJumps*]

See also:

- Options[146].

See also events:

- OnJump[247];
- OnRVMouseMove[254].

See also:

- TRVStyle.SpacesInTab[402];
- RichView Hypertext[35];
- *Tags*[34].

1.4.1.2.1.10 TRichView Edit.ForceFieldHighlight

Allows highlighting label items [101] (including numbered sequences [102], footnotes [104], endnotes [103], sidenotes [105]) even when the editor is not focused.

```
property ForceFieldHighlight: Boolean;
```

(Introduced in version 15)

The editor can highlight label items with Style [156].FieldHighlightColor [390].

There are three modes possible, depending on Style [156].FieldHighlightType [390]: all labels are highlighted, a current label is highlighted, nothing is highlighted.

This property affects the mode when the current (at the caret position) item is highlighted. If *False*, it is highlighted only if the editor has the input focus. If *True*, it is highlighted even in unfocused editors.

This property is useful if you implement editing properties of these items using controls on the same form. It allows to highlight the edited item.

1.4.1.2.1.11 TRichView Edit.LiveSpellingMode

Defines when live spelling thread is started.

```
type
  TRVLiveSpellingMode = (rvlspManualStart, rvlspOnChange);

property LiveSpellingMode: TRVLiveSpellingMode;
```

(introduced in v1.9)

Value	Meaning
rvlspManualStart	Live spelling check can be started only by StartLiveSpelling [237].
rvlspOnChange	Live spelling check also starts when the user modifies document (or when any editing-style method is called).

Default value:

rvlspOnChange

See also:

- Live spelling check in RichView [69].

1.4.1.2.1.12 TRichView Edit.Modified

Value of this property is *True* if document was modified

```
property Modified: Boolean.
```

This is a public (not published) property.

Initially it is *False*. Calling Clear [175] resets it to *False*.

Any editing operation, including calling any editor-style method (which invokes OnChange [357] event) sets it to *True*.

Calling any method introduced in TCustomRichView does not affect it.

If you use this property, set it to *False* after saving document to file.

1.4.1.2.1.13 TRichViewEdit.OffsetInCurItem

Offset of the caret in the current item.

```
property OffsetInCurItem: Integer;
```

Index of item at the position of the caret is returned in CurItemNo[288] property.

CurItemNo and OffsetInCurItem together define position of the caret in the document.

If item is a text item, then caret is before the OffsetInCurItem-th character of string. If caret is after the last character of the text item, OffsetInCurItem = length of the item + 1.

If item is not a text, then

- if the caret is before the item, then OffsetInCurItem=0;
- if the caret is after the item, then OffsetInCurItem=1.

If selection exists, position of the caret is always equal to the second bound of selection (because the caret is always at the end of selection).

You can set the caret position using SetSelectionBounds[235].

If the caret is inside table cell, this property returns 1 (position after the table). Items of subdocuments (table cells[567]) are not included in the items range of the main document. If you want to know position of the caret in the table cell, use TopLevelEditor[295].CurItemNo[288] and TopLevelEditor[295].OffsetInCurItem.

This property must be accessed only when the document is formatted.

See also properties:

- CurItemNo[288];
- TopLevelEditor[295].

See also events:

- OnCaretMove[356].

See also methods:

- SetSelectionBounds[235];
- GetSelectionBounds[203].

See also:

- Inserting in the position of the caret[52];
- How to: move the caret to the beginning or to the end of document in TRichViewEdit[668].

1.4.1.2.1.14 TRichViewEdit.ReadOnly

When set to *True*, this property prevents user's input in the editor; it also blocks execution of all editing-style methods.

```
property ReadOnly: Boolean;
```

This property does not affect viewer-style methods.

Editor hides the caret in read-only mode, if *rvoHideReadOnlyCaret* is in EditorOptions[291].

Since version 1.6, hypertext in read-only mode works automatically (user does not need to press and hold Ctrl key).

Default value:

False

See also events:

- OnChanging[357].

1.4.1.2.1.15 TRichView Edit.SmartPopupProperties

Properties for "smart popups" (analog of Microsoft Office's "smart tags" buttons).

```
property SmartPopupProperties: TRVSmartPopupProperties;
```

(introduced in version 10)

See TRVSmartPopupProperties for details.

See also properties:

- SmartPopupVisible.

See also events:

- OnSmartPopupClick.

1.4.1.2.1.16 TRichView Edit.SmartPopupVisible

Shows/hides "smart popup" button for the current (at the position of caret) item.

```
property SmartPopupVisible: Boolean;
```

(introduced in version 10)

The editor hides this button automatically on document change or clearing.

Since smart popup works only for the current item, you must ensure that it is displayed for the current item: update it in OnCaretMove event. Even if this property is *True*, reassigning *True* to it makes sense, because it links the smart popup to the current item.

Properties for the button are defined in SmartPopupProperties.

See also events:

- OnSmartPopupClick.

1.4.1.2.1.17 TRichView Edit.TopLevelEditor

Returns active editor component

```
property TopLevelEditor: TCustomRichViewEdit;
```

(introduced in version 1.7)

This property returns editor containing the caret.

This is either this RichViewEdit itself or InplaceEditor of table cell which is being edited.

If cell of nested table is being edited, this is InplaceEditor of InplaceEditor, and so on.

Insert*, **GetCurrent***, **SetCurrent*** methods automatically work with content of TopLevelEditor.

Use this property, if you want to work with methods having ItemNo in parameters.

This property must be accessed only when the document is formatted.

1.4.1.2.1.18 TRichView Edit.UndoLimit

Limits capacity of the undo buffer.

```
property UndoLimit: Integer;
```

(Introduced in version 1.3)

Value	Meaning
0	Undo is turned off

-1	Undo buffer "unlimited" (limited by available memory)
any positive value	Limit of saved operations to undo

Note 1: although typing is undone character-by-character, it is saved as a single undo action (for more efficient memory usage).

Note 2: It's not recommended for this version to set UndoLimit to 1. Use multilevel undo instead.

Note 3: Modifying UndoLimit does not affect the current state of undo buffer (no items are deleted from it)

This property does not affect behavior of methods introduced in TRichView, they can't be undone.

Default value:

-1 (unlimited buffer)

See also:

- Undo in RichViewEdit 54.

1.4.1.2.2 Methods

1.4.1.2.2.1 TRichViewEdit.AdjustControlPlacement

Returns the control 94 owned by the **ItemNo**-th item to its position in the document.

```
procedure AdjustControlPlacement(ItemNo: Integer);
```

Usually you do not need to use this method. RichViewEdit moves and scrolls inserted controls when it's needed.

If you want to resize control, use ResizeCurrentControl 335 (or ResizeControl 335) method.

Use this method if the control has moved or resized itself. This method moves the control to the proper place in a document.

If the control has changed its size, this method also quickly reformats the affected part of the document.

It's recommended to use AdjustControlPlacement2 296 instead of this method, because it can work with controls inserted in table 111 cells.

This method must be called only when the document is formatted.

1.4.1.2.2.2 TRichViewEdit.AdjustControlPlacement2

Returns **Control** back to its position in the document. **Control** must be inserted in RichViewEdit directly or in table cell.

```
procedure AdjustControlPlacement2(Control: TControl);
```

(introduced in version 1.4)

Usually you do not need to use this method. RichViewEdit moves and scrolls inserted controls when it's needed.

If you want to resize control, use ResizeCurrentControl 335 (or ResizeControl 335) method.

Use this method if **Control** has moved or resized itself. This method moves the control to the proper place in the document.

If the control has changed its size, this method also quickly reformats the affected part of the document.

This method works only if the control is inserted in the root RichViewEdit or in cell inplace editor.

This method must be called only when the document is formatted.

See also:

- AdjustControlPlacement 296.

1.4.1.2.2.3 TRichView Edit.ApplyListStyle

Applies list style (bullets or numbering) to the selected paragraphs

```
procedure ApplyListStyle(AListNo, AListLevel, AStartFrom: Integer;
  AUseStartFrom, ARecursive: Boolean);
```

(introduced in version 1.7)

This method adds/changes list markers for the selected paragraph(s).

Parameters

AListNo defines style of this list. This is an index in the collection of list styles (Style[156].ListStyles[396])

AListLevel – list level. This is an index in the collection of list levels (Style.ListStyles[**AListNo**].Levels[459]).

If **AListLevel**=-1, levels of the existing markers are not changed, new markers have level=0.

If **AUseStartFrom**=*True*, and this is a numbered list, **AStartFrom** defines value of the list counter for this marker. If **AUseStartFrom**=*False*, the numbering is continued. For bulleted lists, these parameters are ignored.

ARecursive – reserved for future use. Set it to *False*.

Method type: editor-style method (unlike SetListMarkerInfo[233]).

See also:

- RemoveLists[334];
- ChangeListLevels[304].

See also methods of TCustomRichView:

- SetListMarkerInfo[233];
- GetListMarkerInfo[199].

1.4.1.2.2.4 TRichView Edit.ApplyParaStyle

Applies the specified paragraph style.

```
procedure ApplyParaStyle(ParaStyleNo: Integer);
```

ParaStyleNo – index in the collection of paragraph styles (Style[156].ParaStyles[398]).

The method applies the paragraph style **ParaStyleNo** to the paragraph containing the caret and all the selected paragraphs.

If existing paragraphs are formatted with styles having *rvpaoStyleProtect* in their Options[453], this method does not change their styles.

Method type: editing-style.

If you want to implement commands like "change alignment", "increase indent" use ApplyParaStyleConversion[298] instead.

See also properties:

- CurParaStyleNo[289].

See also methods:

- ApplyTextStyle[300];
- ApplyParaStyleConversion[298].

See also:

- TParaInfo.Options[453];
- Paragraphs overview[37].

1.4.1.2.2.5 TRichView Edit.ApplyParaStyleConversion

Applies a custom conversion procedure to styles of the selected paragraphs.

```
procedure ApplyParaStyleConversion(UserData: Integer; Recursive: Boolean = True);
```

(introduced in version 1.7)

This procedure calls OnParaStyleConversion[364] event for each selected item. Using this method, you can implement commands like "change alignment" or "increase indent".

This method ignores any paragraph protection[453], it can change paragraph style even if *rvpaoStyleProtect* is included in Options[453] of existing paragraphs' styles. If you want to respect paragraph style protection, check this option yourself in OnParaStyleConversion[364] event.

If **Recursive** = *False*, OnParaStyleConversion[364] is not called for cells of selected tables[98] (except for the case of multicell selection in a single table).

Method type: editing-style.

Demo:

- Demos\Delphi\Editors\Editor 2\
- Demos\CBuilder\Editors\Editor 2\

See also:

- ApplyStyleConversion[299];
- ApplyParaStyle[297].

See also events:

- OnParaStyleConversion[364].

See also:

- Paragraphs overview[37].

1.4.1.2.2.6 TRichView Edit.ApplyParaStyleTemplate

Applies the specified style template to the selected paragraphs.

```
procedure ApplyParaStyleTemplate(TemplateNo: Integer;
  ResetAdditionalFormatting: Boolean; Recursive: Boolean=True);
```

(introduced in version 14)

This procedure assigns the specified style template to paragraph styles[398] of items in the selected paragraphs.

Parameters:

TemplateNo – index in the collection Style[156].StyleTemplates[402], or the value -1. The value -1 means a format clearing (see below). The style template must have Kind[461] = *rvstkParaText* or *rvstkPara*.

If **ResetAdditionalFormatting**=*True*, all properties of the specified style template will be applied. If **ResetAdditionalFormatting**=*False*, only "unmodified" properties (see below) will be changed. This parameter is ignored if **TemplateNo**=-1.

If **Recursive** = *False*, this method does not change styles in cells of selected tables[98] (except for the case of multicell selection in a single table).

This method does nothing if UseStyleTemplates[158]=*False* and **TemplateNo**>=0.

If paragraphs are formatted with styles having *rvpaoStyleProtect* in their Options[453], this method skips them.

Format clearing

A format clearing means the following procedure. If "Normal" style template exists, the method works like applying this style template with **ResetAdditionalFormatting**=*True*. If it does not exist, the method resets all properties of paragraph styles to default values, and assigns StyleTemplateId[432]=-1.

If UseStyleTemplates[158]=*False* and **TemplateNo**<0, the results are identical to calling ApplyParaStyle[297](0).

Modified properties

Properties of a paragraph style[456] are considered modified, if they have values different from values of the corresponding properties of the current style template (referred in StyleTemplateId[432] property). If a style template is not assigned, the properties are considered modified if they have non-default values. The method ignores ModifiedProperties[457] property (it calculates this property itself).

Method type: editing-style.

See also:

- ApplyTextStyleTemplate[301];
- ApplyParaStyleConversion[298];
- ApplyParaStyle[297].

1.4.1.2.2.7 TRichView Edit.ApplyStyleConversion

Applies a custom conversion procedure to styles of the selected text items[88], tabs[89], label items[101] (and item of all types inherited from label items).

```
procedure ApplyStyleConversion(UserData: Integer; Recursive: Boolean = True);
```

This procedure calls OnStyleConversion[366] event for each selected text item and for the current text style[290]. Using this method, you can implement commands like "make bold" or "change font name".

This method ignores any text protection[439], it can change text style even if *rvprStyleProtect* is included in Protection[439] of existing text items' styles. If you want to respect text style protection, check this protection option yourself in OnStyleConversion[366] event.

If **Recursive** = *False*, OnStyleConversion[366] is not called for cells of selected tables[98] (except for the case of multicell selection in a single table).

Method type: editing-style.

Unicode note: If Unicode[446] text style is applied to ANSI text item, text in this item is converted from ANSI to Unicode, and vice versa.

Demo:

- Demos\Delphi\Editors\Editor 2\
- Demos\CBuilder\Editors\Editor 2\

See also:

- ApplyParaStyleConversion[298];
- ApplyTextStyle[300].

See also events:

- OnStyleConversion[366].

1.4.1.2.2.8 TRichView Edit.ApplyStyleTemplate

Applies the specified style template to the selected text or paragraphs.

```
procedure ApplyStyleTemplate(TemplateNo: Integer; Recursive: Boolean=True);
```

(introduced in version 14)

This procedure assigns the specified style template to text styles[404] or paragraph styles[398] of items in the selected paragraphs. The result depends on the style template's Kind[461] and the selection.

Parameters:

TemplateNo – index in the collection Style[156].StyleTemplates[402], or the value -1. The value -1 means a format clearing (see below).

If **Recursive** = *False*, this method does not change styles in cells of selected tables [98] (except for the case of multicell selection in a single table).

This method does nothing if UseStyleTemplates [158]=*False* and **TemplateNo**>=0.

The method works according to the following rules:

- if Kind [461] of the specified style template = *rvstkText*, it is applied to the selected text;
- if Kind [461] of the specified style template = *rvstkPara*, it is applied to the selected paragraphs; the only exception is "Normal" style template [428];
- if **TemplateNo**=-1, or Kind [461] of the specified style template = *rvstkParaText*, or this is a "Normal" style template, the result depends on the selection. If nothing is selected, or the whole paragraph is selected, or the selection includes multiple paragraphs/cells, the method applies to the selected paragraphs. Otherwise, it applies to the selected text.

The method clears additional formatting, like ApplyTextStyleTemplate [301] or ApplyParaStyleTemplate [298] called with **ResetAdditionalFormatting**=*True*.

The method skips paragraph styles having *rvpaoStyleProtect* in their Options [453], and text styles having *rvprStyleProtect* in their Protection [439].

Format clearing

The method clears text or paragraph formatting in two cases: when it is called with **TemplateNo**=-1, or when it applies "Normal" style template. The results of format clearing is described in the topics about ApplyTextStyleTemplate [301] and ApplyParaStyleTemplate [298].

Method type: editing-style.

See also:

- ChangeStyleTemplates [304].

1.4.1.2.2.9 TRichView Edit.ApplyTextStyle

Applies the specified text style to the selected items (text items [88], tabs [89], label items [101])

```
procedure ApplyTextStyle(TextStyleNo: Integer);
```

TextStyleNo – index in the collection of text styles (Style [156].TextStyles [404]).

If selection is empty, this method only sets CurTextStyleNo [290] to **TextStyleNo**. If the selection is not empty, the method changes style of text items [88] in the selection.

If the existing text items are formatted with styles having *rvprStyleProtect* in their Protection [439], this method does not change their styles.

Method type: editing-style.

Unicode note: If Unicode [446] text style is applied to ANSI text item, text in this item is converted from ANSI to Unicode, and vice versa.

If you want to implement commands like "make bold", "change text color", "apply font", use ApplyStyleConversion [299] instead of this method.

See also properties:

- CurTextStyleNo [290];
- EditorOptions [291].

See also methods:

- ApplyParaStyle [297].

1.4.1.2.2.10 TRichView Edit.ApplyTextStyleTemplate

Applies the specified style template to the selected text items [88], tabs [89], label items [101] (and to items of all types inherited from label items).

```
procedure ApplyTextStyleTemplate(TemplateNo: Integer;
  ResetAdditionalFormatting: Boolean; Recursive: Boolean=True);
```

(introduced in version 14)

This procedure assigns the specified style template to text styles [404] of the selected items.

Parameters:

TemplateNo – index in the collection Style [156].StyleTemplates [402], or the value -1. The value -1 means a format clearing (see below). The style template must have Kind [461] = *rvstkParaText* or *rvstkText*. The only exception is "Normal" style template [428]: applying it clears text format as well, but **ResetAdditionalFormatting** is respected.

If **ResetAdditionalFormatting**=*True*, all properties of the specified style template will be applied. If **ResetAdditionalFormatting**=*False*, only "unmodified" properties (see below) will be changed. This parameter is ignored if **TemplateNo**=-1.

If **Recursive** = *False*, this method does not change styles in cells of selected tables [98] (except for the case of multicell selection in a single table).

This method does nothing if UseStyleTemplates [158]=*False* and **TemplateNo**>=0.

If items are formatted with text styles having *rvprStyleProtect* in their Protection [439], this method skips them.

Format clearing

A format clearing means resetting all text attributes to default values. If a paragraph style template (ParaStyleTemplateId [439]) is assigned, it defines these default values. A special processing is performed for hyperlinks (i.e. text styles with Jump [445]=*True*): if "Hyperlink" style template exists, it is applied (in other words, the method works like applying this style template with **ResetAdditionalFormatting**=*True*).

If UseStyleTemplates [158]=*False* and **TemplateNo**<0, the results are identical to calling ApplyTextStyle [300](0).

Modified properties

Properties of a text style [444] are considered modified, if they have values different from values of the corresponding properties of the current style templates (referred in StyleTemplateId [432] and ParaStyleTemplateId [439] properties). If none of these style templates are assigned, the properties are considered modified of they have non-default values. The method ignores ModifiedProperties [445] property (it calculates this property itself).

Method type: editing-style.

See also:

- ApplyParaStyleTemplate [298];
- ApplyStyleConversion [299];
- ApplyTextStyle [300].

1.4.1.2.2.11 TRichView Edit.BeginItemModify

Stores some information required for quick reformatting after modification of item.

```
procedure BeginItemModify(ItemNo: Integer; var ModifyData: Integer);
```

Input parameter:

ItemNo – index of item which will be modified.

Output parameter:

ModifyData receives some formatting information. Value assigned to this parameter should be passed to

EndItemModify 307 after performing modifications.

Usually this function is used when modifying tables 528.

This method must be called only when the document is formatted.

Example is in Table Operations Overview 119

1.4.1.2.2.12 TRichView Edit.BeginUndoCustomGroup

Begins a new undo action.

```
procedure BeginUndoCustomGroup(const Name: String);
```

Begins a new undo action.

Usually followed by SetUndoGroupMode 355(*True*) ... SetUndoGroupMode(*False*), thus allowing to group several subsequent changes in editor (they will be undone/redone as one operation).

The same as BeginUndoGroup 302, but allows to give a custom name to this group of changes.

This method must be called only when the document is formatted.

This method must be called directly for the editor where changed are made (usually TopLevelEditor 295).

See also:

- Undo in RichViewEdit 54.

1.4.1.2.2.13 TRichView Edit.BeginUndoGroup

Begins a new undo action.

```
procedure BeginUndoGroup(UndoType: TRVUndoType 647);
```

Begins a new undo action.

Usually followed by SetUndoGroupMode 355(*True*) ... SetUndoGroupMode(*False*), thus allowing to group several subsequent changes in editor (they will be undone/redone as one operation).

The same as BeginUndoCustomGroup 302, but gives a standard name (undo identifier) to this group of changes.

This method must be called only when the document is formatted.

This method must be called directly for the editor where changed are made (usually TopLevelEditor 295).

See also:

- Undo in RichViewEdit 54, includes example.

1.4.1.2.2.14 TRichView Edit.BeginUpdate

Suspends repainting until EndUpdate 307 is called.

```
procedure BeginUpdate;
```

(meaning of this method is changed since version 10)

Use BeginUpdate to speed processing and avoid flicker while multiple editing operations are applied.

Calls to BeginUpdate and EndUpdate can be nested.

1.4.1.2.2.15 TRichView Edit.CanChange

Returns: "can the document in this editor be changed now?"

```
function CanChange: Boolean;
```

This function checks ReadOnly 294 property.

Usually this function is called before performing operations on tables 528 to prevent modifying editor in read-only

state.

For TDBRichViewEdit[377], this function also turns the editor's dataset in editing mode (required before operations on tables).

This method must be called only when the document is formatted.

1.4.1.2.2.16 TRichView Edit.CanPaste

Returns *True* if:

- the Clipboard contains data in at least one format which RichViewEdit can paste (and this format is included in AcceptPasteFormats[287] property), and
- the editor is not in read-only state (ReadOnly[294]=*False*)

```
function CanPaste: Boolean;
```

See also methods:

- Paste[328];
- CanPasteRVF[303];
- CanPasteRTF[303].

See also:

- Working with Clipboard[48].

1.4.1.2.2.17 TRichView Edit.CanPasteRTF

Returns *True* if:

- the Clipboard contains data in RTF (Rich Text Format), and
- the editor is not in read-only state (ReadOnly[294]=*False*)

```
function CanPasteRTF: Boolean;
```

(Introduced in version 1.5)

See also methods:

- PasteRTF[330];
- CanPaste[303].

See also:

- Working with Clipboard[48]

1.4.1.2.2.18 TRichView Edit.CanPasteRVF

Returns *True* if:

- the Clipboard contains data in RVF (RichView Format[59]), and
- the editor is not in read-only state (ReadOnly[294]=*False*)

```
function CanPasteRVF: Boolean;
```

See also methods:

- PasteRVF[331];
- CanPaste[303].

See also:

- Working with Clipboard[48];
- RVF[59].

1.4.1.2.2.19 TRichView Edit.Change

Generates OnChange[357] event.

```
procedure Change;
```

This method also assigns *True* to Modified[293] property.

1.4.1.2.2.20 TRichView Edit.ChangeListLevels

Changes list levels (levels of paragraphs bullets and numbering) for the selected paragraphs.

```
procedure ChangeListLevels(LevelDelta: Integer);
```

(introduced in version 1.7)

For example,

```
  rve.ChangeListLevels(+1) // increases level by 1
  rve.ChangeListLevels(-1) // decreases level by 1
```

Paragraphs without bullets or numbering are not affected.

Method type: editing-style.

See also:

- ApplyListStyle[297];
- RemoveLists[334].

See also methods of TCustomRichView:

- SetListMarkerInfo[233];
- GetListMarkerInfo[199].

1.4.1.2.2.21 TRichView Edit.ChangeStyleTemplates

Updates the document after editing style templates.

```
procedure ChangeStyleTemplates(NewStyleTemplates: TRVStyleTemplateCollection[427]);
```

(introduced in version 14)

The code below shows how to edit style templates for RichViewEdit1: TRichViewEdit. It assumes that you have RVStyleTmp: TRVStyle[385] component, not linked to any RichView.

```
RVStyleTmp.Units[404] := RichViewEdit1.Style.Units;
RVStyleTmp.StyleTemplates[402].AssignStyleTemplates[428](RichViewEdit1.Style.StyleTemplates, Tru
<code for changing RVStyleTmp.StyleTemplates here>
RichViewEdit1.ChangeStyleTemplates(RVStyleTmp.StyleTemplates);
```

While editing RVStyleTmp.StyleTemplates, you can:

- add new style templates;
- change any property of style templates, rename them;
- delete style templates (use RVStyleTmp.StyleTemplates.ForbiddenIds[428]).

Method type: editing-style.

ScaleRichView note: in TSRichViewEdit, this method must be called for SRichViewEdit.ActiveEditor. It may be called while the main editor, a header or a footer is being edited. It cannot be called while a footnote or an endnote is being edited.

See also:

- ApplyStyleTemplate[299];
- ApplyParaStyleTemplate[298];
- ApplyTextStyleTemplate[301].

1.4.1.2.2.22 TRichView Edit.Clear

Clears the document.

```
procedure Clear;override;
```

This method:

- deletes all content of the document (see RichView.Clear[175]);
- sets the current text style index (CurTextStyleNo[290]) and the current paragraph style index (CurParaStyleNo[289]) to zeros;

- clears undo and redo buffers (see ClearUndo[305]);
- sets Modified[293] to *False*.

This method does not check ReadOnly[294], does not reformat and repaint component, does not generate OnChange[357].

This methods makes the document unformatted.

1.4.1.2.2.23 TRichView Edit.ClearTextFlow

Clears text flow around left- and right-aligned items.

```
procedure ClearTextFlow(Left, Right: Boolean);
```

(introduced in version 12)

This method changes ClearLeft[138] and ClearRight[138] property for the first selected paragraph.

Method type: editing-style.

See also:

- TRVVAlign[648] type (*rvvaLeft* and *rvvaRight*)

1.4.1.2.2.24 TRichView Edit.ClearUndo

Clears undo and redo buffers of the editor.

```
procedure ClearUndo;
```

(Introduced in version 1.3)

See also:

- Undo in RichViewEdit[54].

1.4.1.2.2.25 TRichView Edit.ConvertToHotPicture

Converts the item type from picture[90] to *hot-picture*[93].

```
procedure ConvertToHotPicture(ItemNo: Integer);
```

Hot-picture is a picture with hypertext link.

If **ItemNo**<0, this methods works with the item at the position of caret. Otherwise (if **ItemNo** in range 0..ItemCount[144]-1) it works with the specified item

If this item is not a picture (GetItemStyle[194]/CurItemStyle[289] <> *rvsPicture*), this method does nothing.

Method type: editing-style.

See also:

- ConvertToPicture[305].

1.4.1.2.2.26 TRichView Edit.ConvertToPicture

Converts the item type from *hot-picture*[93] to picture[90]

```
procedure ConvertToPicture(ItemNo: Integer);
```

Hot-picture is a picture with hypertext link.

If **ItemNo**<0, this methods works with the item at the position of caret. Otherwise (if **ItemNo** in range 0..ItemCount[144]-1) it works with the specified item.

If this item is not a *hot-picture* (GetItemStyle[194]/CurItemStyle[289] <> *rvsHotPicture*), this method does nothing.

Method type: editing-style.

See also:

- ConvertToHotPicture[305].

1.4.1.2.2.27 TRichView Edit.Create

Usual component constructor.

Creates a new RichViewEdit instance.

```
constructor Create(AOwner: TComponent);override;
```

Use Create to instantiate RichViewEdit at runtime. RichViewEdits placed on forms at design time are created automatically. Specify the owner of the new RichViewEdit using the AOwner parameter.

Important note: initial values of several properties of components created in code may be different from initial values of components placed on form at design time. In the latter case, initial values of these properties can be overridden by the designtime component editor[134] (and overridden by default!)

1.4.1.2.2.28 TRichView Edit.CutDef

Cuts selection to the Clipboard

```
procedure CutDef;
```

This method copies selection to the Clipboard using CopyDef[177] method. If the selection was copied (at least one of copy formats is selected in Options[146]), it calls DeleteSelection[306].

This method is executed automatically when user presses Ctrl+X or Shift+Delete.
Method type: editing-style.

See also methods:

- CopyDef[177];
- Paste[328];
- SelectionExists[224].

See also:

- Working with selection[47];
- Working with Clipboard[48].

1.4.1.2.2.29 TRichView Edit.DeleteSelection

Deletes the selected fragment in the editor.

```
procedure DeleteSelection;
```

Method type: editing-style.

See also:

- Working with selection[47].

1.4.1.2.2.30 TRichView Edit.Destroy

Destroys an instance of TRichViewEdit

```
destructor Destroy;override;
```

Do not call Destroy directly in an application. Instead, call Free. Free verifies that the control is not *nil*, and only then calls Destroy.

Applications should only free controls explicitly when the constructor was called without assigning an owner to the control.

1.4.1.2.2.31 TRichView Edit.EndItemModify

Quickly reformats editor after modifying item.

```
procedure EndItemModify(ItemNo: Integer; ModifyData: Integer);
```

Parameters:

ItemNo – index of item which was modified.

ModifyData – the value obtained in BeginItemModify [301]

Usually this function is used when modifying tables [528].

Example is in Table Operations Overview [119]

1.4.1.2.2.32 TRichView Edit.EndUpdate

Reenables editor repainting that was turned off with the BeginUpdate [302] method. Repaints the editor.

```
procedure EndUpdate;
```

(meaning of this method is changed since v10)

Calls to BeginUpdate are cumulative, so calling EndUpdate will reenable repainting only if every call to BeginUpdate has been matched by a call to EndUpdate.

1.4.1.2.2.33 TRichView Edit.GetCheckpointAtCaret

Returns *checkpoint* at the position of caret.

```
function GetCheckpointAtCaret: TCheckpointData;
```

(introduced in version 10)

This method returns *checkpoint* only if the caret is at the beginning of item, otherwise it returns *nil*.

The following table compares this method with GetCurrentCheckpoint [309].

Caret Position	GetCheckpointAtCaret	GetCurrentCheckpoint [309]
In the middle of text item	returns *nil*	returns *checkpoint* for this item
At the beginning of item and at the beginning of line	returns *checkpoint* for this item	returns *checkpoint* for this item
At the beginning of item and in the middle of line	returns *checkpoint* for this item	returns *checkpoint* for the previous item
At the end of item and at the end of line	returns *nil*	returns *checkpoint* for this item

This method is used in conjunction with InsertCheckpoint [318] and RemoveCheckpointAtCaret [333]. The alternative set of methods is: SetCurrentCheckpointInfo [343], GetCurrentCheckpoint [309] and RemoveCurrentCheckpoint [334].

See also:

- "Checkpoints" [30].

1.4.1.2.2.34 TRichView Edit.GetCurrentBreakInfo

Returns main properties for the item of *break* 93 type at the position of caret.

```
procedure GetCurrentBreakInfo(var AWidth: TRVStyleLength644;
  var AStyle: TRVBreakStyle624; var AColor: TColor;
  var ATag: TRVTag645);
```

GetCurrentBreakInfo(...) is equivalent to TopLevelEditor295.**GetBreakInfo**183(TopLevelEditor295.CurItemNo288, ...).

The item must be of *break* 93 type (*rvsBreak* 663), otherwise the method raises ERichViewError608 exception. Type of item at the position of caret is returned by CurItemStyle289.

Output parameters:

AWidth – line width (or rectangle height). This value is measured in Style156.Units404.

AStyle – visual style of this *break*, see TRVBreakStyle624 for possible values.

AColor – line color. If it is equal to *clNone*, Style156.TextStyles404[0].Color435 is used.

ATag – *tag* of the item. Use SetCurrentTag349 or SetCurrentBreakInfo342 to change *tag* of item as an editing operation.

Instead of this method, you can use GetCurrentItemExtraIntPropertyEx312 and GetCurrentTag316 methods.

This method must be called only when the document is formatted.

Additional properties of item at the position of caret are returned by the methods GetCurrentItemExtraIntProperty312 and GetCurrentItemExtraStrProperty313.

See also methods:

- SetCurrentBreakInfo342 (changes properties of *break* at the position of caret, as an editing operation);
- GetBreakInfo183 (returns properties of the specified *break*).

See also properties:

- CurItemNo288;
- CurItemStyle289.

See also:

- Obtaining RichView items50;
- Item types29;
- "Tags"34.

1.4.1.2.2.35 TRichView Edit.GetCurrentBulletInfo

Returns main properties for the item of *bullet* 96 type at the position of caret.

```
procedure GetCurrentBulletInfo(var AName: TRVAnsiString623;
  var AImageIndex: Integer; var AImageList: TCustomImageList;
  var ATag: TRVTag645);
```

GetCurrentBulletInfo(...) is equivalent to TopLevelEditor295.**GetBulletInfo**184(TopLevelEditor295.CurItemNo288, ...).

Output parameters:

AName – name of *bullet* 96. It can also be read using GetCurrentItemText313 method.

AImageList – image list (a reference to image list, do not free it).

AImageIndex – index in **AImageList**. It can also be read using GetCurrentItemExtraIntPropertyEx312 method.

ATag – *tag* of the item. Use SetCurrentTag349 or SetCurrentBulletInfo342 to change *tag* of item as an editing operation. This value can also be read using GetCurrentTag316 method.

This method must be called only when the document is formatted.

Unicode note: names of non-text items are ANSI strings.

Additional properties of item at the position of caret are returned by the methods GetCurrentItemExtraIntProperty 312 and GetCurrentItemExtraStrProperty 313.

See also methods:

- SetCurrentBulletInfo 342 (changes properties of *bullet* at the position of caret, as an editing operation);
- GetBulletInfo 184 (returns properties of the specified *bullet*).

See also properties:

- CurItemNo 288;
- CurItemStyle 289.

See also:

- Obtaining RichView items 50;
- Item types 29;
- "Tags" 34.

1.4.1.2.2.36 TRichView Edit.GetCurrentCheckpoint

Returns *checkpoint* associated with the item at position of caret, or *nil* if this item does not have associated *checkpoint*.

```
function GetCurrentCheckpoint: TCheckPointData 622;
```

The same as TopLevelEditor 295.**GetItemCheckpoint** 191(TopLevelEditor 295.CurItemNo 288).

This method must be called only when the document is formatted.

See also methods:

- SetCurrentCheckpointInfo 343 (changes *checkpoint* of item at the position of caret, as an editing operation);
- GetItemCheckpoint 191 (returns *checkpoint* of the specified item).

See also properties:

- CurItemNo 288.

See also:

- "Checkpoints" 30.

1.4.1.2.2.37 TRichView Edit.GetCurrentControlInfo

Returns main properties for the item of inserted control 94 type at position of caret.

```
procedure GetCurrentControlInfo(var AName: TRVAnsiString 623;
  var Actrl: TControl; var AVAlign: TRVVAlign 648;
  var ATag: TRVTag 645);
```

GetCurrentControlInfo(...) is equivalent to TopLevelEditor 295.**GetControlInfo** 187(TopLevelEditor 295.CurItemNo 288, ...).

Output parameters:

AName – name of control item 94. Do not confuse with **Actrl**.Name property! This value can also be read using GetCurrentItemText 313 method.

Actr – the control itself. This method returns control owned by RichView, do not not destroy it.

AVAlign – vertical alignment of the control.

ATag – *tag* of the item. Do not confuse with **Actrl**.Tag property!. Use SetCurrentTag 349 or SetCurrentControlInfo 344 to change *tag* of item as an editing operation. This value can also be read using GetCurrentTag 316 method.

This method must be called only when the document is formatted.

Unicode note: names of non-text items are ANSI strings.

Additional properties of item at the position of caret are returned by the methods GetCurrentItemExtraIntProperty 312 and GetCurrentItemExtraStrProperty 313.

See also methods:

- SetCurrentControlInfo 344 (changes properties of control item at the position of caret, as an editing operation);
- GetControlInfo 187 (returns properties of the specified control item).

See also properties:

- CurItemNo 288;
- CurItemStyle 289.

See also:

- Obtaining RichView items 50;
- Item types 29;
- "Tags" 34.

1.4.1.2.2.38 TRichView Edit.GetCurrentHotspotInfo

Returns main properties for the item of *hotspot* 97 type at the position of caret.

```
procedure GetCurrentHotspotInfo(var AName: TRVAnsiString 623;
  var AImageIndex, AHotImageIndex: Integer;
  var AImageList: TCustomImageList;
  var ATag: TRVTag 645);
```

GetCurrentHotspotInfo(...) is equivalent to TopLevelEditor 295.**GetHotspotInfo** 189(TopLevelEditor 295.CurItemNo 288, ...).

Output parameters:

AName – name of *hotspot* 97. It can also be read using GetCurrentItemText 313 method.

AImageList – image list (a reference to image list, do not free it).

AImageIndex – index in **AImageList** for normal image. It can also be read using GetCurrentItemExtraIntPropertyEx 312 method.

AHotImageIndex – index in **AImageList** for "hot" image. This image is displayed under the mouse pointer (in hypertext mode), or when user moves the caret to this item (in TRichViewEdit). It can also be read using GetCurrentItemExtraIntPropertyEx 312 method.

ATag – *tag* of the item. Use SetCurrentTag 349 or SetCurrentHotspotInfo 345 to change *tag* of item as an editing operation. This value can also be read using GetCurrentTag 316 method.

This method must be called only when the document is formatted.

Unicode note: names of non-text items are ANSI strings.

Additional properties of item at the position of caret are returned by the methods GetCurrentItemExtraIntProperty 312 and GetCurrentItemExtraStrProperty 313.

See also methods:

- SetCurrentHotspotInfo 345 (changes properties of *hotspot* at the position of caret, as an editing operation);
- GetHotspotInfo 189 (returns properties of the specified *hotspot*).

See also properties:

- CurItemNo 288;
- CurItemStyle 289.

See also:

- Obtaining RichView items 50;
- Item types 29;
- "Tags" 34.

1.4.1.2.2.39 TRichView Edit.GetCurrentItem

Returns object representing item at position of caret.

```
function GetCurrentItem: TCustomRVItemInfo 526;
```

(introduced in version 1.4)

This method is usually used when working with tables 98, labels 101, sequences 102, footnotes 104, endnotes 103, references to notes 108.

The table object is returned if the caret is to the left or to the right of the table, or if several table cells are selected. If the caret is inside table cell, this method returns item contained in this cell. To get the table in this case, use GetCurrentItemEx 311 instead of GetCurrentItem.

This method must be called only when the document is formatted.

When called for an empty editor (i.e. ItemCount 144=0), the method returns *nil*.

See also methods:

- GetCurrentItemEx 311;
- InsertItem 320;
- GetItem 190.

See also properties:

- CurItemNo 288;
- CurItemStyle 289.

See also:

- Obtaining RichView items 50;
- Tables in RichView 111.

1.4.1.2.2.40 TRichView Edit.GetCurrentItemEx

Returns object at the position of caret representing item of the given class.

```
function GetCurrentItemEx(RequiredClass: TCustomRVItemInfoClass 526;
  var ItemRichViewEdit: TCustomRichViewEdit;
  var Item: TCustomRVItemInfo 526): Boolean;
```

(introduced in version 1.4)

This method is usually used when working with tables 111.

Input parameter:

RequiredClass – the required class of item, set it to TRVTableItemInfo 528.

Output parameters:

ItemRichViewEdit – editor containing this item (table); it can be the main editor, or cell inplace-editor (if this item (table) is in cell of another table).

Item – object representing the item (table)

This method must be called only when the document is formatted.

Return value

The method returns *True* if the output parameters were assigned. In case of tables, it returns *True* if:

- the caret is to the right or to the left of the table, or
- the table has multicell selection, or
- the caret is inside table cell.

In case of nested tables, this method returns the top level table (the closest to the user in Z-order).

See also methods:

- GetCurrentItem 311;
- InsertItem 320;
- GetItem 190.

See also:

- Obtaining RichView items 50;
- Tables in RichView 111.

1.4.1.2.2.41 TRichView Edit.GetCurrentItemExtraIntProperty -Ex

The methods return a value of the specified integer property of the item at the position of caret

```
function GetCurrentItemExtraIntProperty(
  Prop: TRVExtraItemProperty; var Value: Integer): Boolean;
function GetCurrentItemExtraIntPropertyEx(
  Prop: Integer; var Value: Integer): Boolean;
```

(introduced in versions 1.7 and 15)

GetCurrentItemExtraIntProperty[Ex](...) is equivalent to TopLevelEditor 295.**GetItemExtraIntProperty[Ex]** 193 (TopLevelEditor 295.CurItemNo 288, ...).

Value receives value of the property identified by **Prop**.

In **GetCurrentItemExtraIntProperty**, **Prop**'s type is TRVExtraItemProperty 627. See information about this type for the list of properties.

In **GetCurrentItemExtraIntPropertyEx**, **Prop**'s type is Integer. If **Prop** can be converted to TRVExtraItemProperty, **GetCurrentItemExtraIntPropertyEx** works like **GetCurrentItemExtraIntProperty**. In addition, it supports properties identified by rveipc*** constants 658

These methods must be called only when the document is formatted.

Return value

True, if this item has this property. *False*, if not.

See also

- SetCurrentItemExtraIntProperty[Ex] 345 (set a value of an integer property for the item at the position of caret, as an editing operation);
- SetItemExtraIntProperty[Ex]Ed 350 (set a value of an integer property for the specified item, as an editing operation);
- GetCurrentItemExtra 313**Str** 313Property 313 (return a value of the specified *string* property of the item at the position of caret).

See also methods of TRichView

- GetItemExtraIntProperty[Ex] 193.

1.4.1.2.2.42 TRichView Edit.GetCurrentItemExtraStrProperty -Ex

The methods return a value of the specified string property of the item at the position of caret

```
function GetCurrentItemExtraStrProperty(
  Prop: TRVExtraItemStrProperty; var Value: String): Boolean;
function GetCurrentItemExtraStrProperty(
  Prop: Integer; var Value: String): Boolean;
```

(introduced in versions 1.9 and 15)

GetCurrentItemExtraStrProperty[Ex](...) is equivalent to TopLevelEditor.**GetItemExtraStrProperty[Ex]**(TopLevelEditor.CurItemNo, ...).

Value receives value of the property identified by **Prop**.

In **GetCurrentItemExtraStrProperty**, **Prop**'s type is TRVExtraItemStrProperty. See information about this type for the list of properties.

In **GetCurrentItemExtraStrPropertyEx**, **Prop**'s type is Integer. If **Prop** can be converted to TRVExtraItemStrProperty, **GetCurrentItemExtraStrPropertyEx** works like **GetCurrentItemExtraStrProperty**. In addition, it supports properties identified by rvespc*** constants

This method must be called only when the document is formatted.

Return value

True, if this item has this property. *False*, if not.

Unicode note: string properties are Unicode strings for Delphi/C++Builder 2009 or newer, and ANSI strings for older versions of Delphi/C++Builder.

See also:

- SetCurrentItemExtraStrProperty[Ex] (set a value of a string property for the item at the position of caret, as an editing operation);
- SetItemExtraStrProperty[Ex]Ed (set a value of a string property for the specified item, as an editing operation);
- GetCurrentItemExtra**Int**Property[Ex] (return a value of the specified *integer* property of the item at the position of caret).

See also methods of TRichView:

- GetItemExtraStrProperty[Ex].

1.4.1.2.2.43 TRichView Edit.GetCurrentItemText -A -W

Returns text for the item at the position of caret.

```
function GetCurrentItemText: String;
function GetCurrentItemTextA: TRVAnsiString;
function GetCurrentItemTextW: TRVUnicodeString;
```

(Introduced in version 1.4, 1.7)

For text items, these methods return visible text. For non-text items, they return item name.

- **GetCurrentItemText** is the same as TopLevelEditor.**GetItemText**(TopLevelEditor.CurItemNo).
- **GetCurrentItemTextA** is the same as TopLevelEditor.**GetItemTextA**(TopLevelEditor.CurItemNo).
- **GetCurrentItemTextW** is the same as TopLevelEditor.**GetItemTextW**(TopLevelEditor.CurItemNo).

Unicode notes:

GetCurrentItemTextA always returns ANSI string:

- If the item is a non-text item, or a text item of ANSI style, the text is returned as it is.
- If the item is a Unicode text item, the function converts Unicode text to ANSI (Unicode text is converted basing on Charset[434] of the item text style).

GetCurrentItemTextW always returns Unicode string:

- If the item is a Unicode text item, the text is returned as it is.
- If the item is a non-text item, or a text item of ANSI style, the function converts ANSI text to Unicode (ANSI text is converted basing on Charset[434] of the item text style, or on RVStyle.DefCodePage[388] for non-text items).

GetCurrentItemText works like **GetCurrentItemTextW** in Delphi/C++Builder 2009 or newer, and like **GetCurrentItemTextA** in the older versions of Delphi/C++Builder.

These methods must be called only when the document is formatted.

See also methods:

- SetCurrentItemText -A -W[347].

See also methods of TCustomRichView:

- GetItemText -A -W[195].

See also properties:

- CurItemNo[288];
- CurItemStyle[289].

See also:

- Obtaining RichView items[50];
- Item types[29];
- "Tags"[34];
- Unicode in RichView[64].

1.4.1.2.2.44 TRichView Edit.GetCurrentItemVAlign

Returns vertical alignment (position relative to the line) of the item at the caret position

```
function GetCurrentItemVAlign: TRVVAlign[648];
```

(introduced in version 12)

The same as TopLevelEditor[295].**GetItemVAlign**[196](TopLevelEditor[295].CurItemNo[288]).

This method must be called only when the document is formatted.

This method must not be called if the current item is one of: text items[88], tables[98], *breaks*[93], tabs[89], list markers[99], footnotes[104] and endnotes[103].

See also methods:

- SetCurrentItemVAlign[348].

See also properties:

- CurItemNo[288].

1.4.1.2.2.45 TRichView Edit.GetCurrentLineCol

Returns a line and column at the caret position.

```
procedure GetCurrentLineCol(var Line, Column: Integer);
```

(introduced in version 1.7)

Number of lines depends on word-wrapping.

Number of the first line is 1. Number of the leftmost caret position is 1.

Lines in table cells are calculated relative to the beginning of the cell.

Warning: this function contains calculations inside, and it may take some time to execute it

(There is an idea how to make it lightning-fast in future, but it will be implemented later)

This method must be called only when the document is formatted.

See also methods of TCustomRichView:

- GetLineNo 198.

1.4.1.2.2.46 TRichView Edit.GetCurrentMisspelling

Returns misspelled word in the position of caret, optionally selects it.

```
function GetCurrentMisspelling(SelectIt: Boolean;
  var Word: String; var StyleNo: Integer): Boolean;
```

Input parameters:

SelectIt – if *True*, the returned word will be selected in the editor.

Output parameters:

Word receives misspelled word.

StyleNo receives text style of misspelled word (index in TextStyles 404 collection of the linked 156 TRVStyle component).

The output parameters are defined only if this method returns *True*.

Unicode note: Word parameter is Unicode string for Delphi/C++Builder 2009, and ANSI string for the older versions of Delphi/C++Builder.

Return value

True if misspelling is found.

See also:

- Live spelling check in RichView 69.

1.4.1.2.2.47 TRichView Edit.GetCurrentPictureInfo

Returns main properties for the item of picture 90/*hot-picture* 93 type at the position of caret.

```
procedure GetCurrentPictureInfo(var AName: TRVAnsiString 623; var Agr: TGraphic;
  var AVAlign: TRVVAlign 648; var ATag: TRVTag 645);
```

GetCurrentPictureInfo(...) is equivalent to TopLevelEditor 295.**GetPictureInfo** 201(TopLevelEditor 295.CurItemNo 288, ...).

Output parameters:

AName – name of the item. It can also be read using GetCurrentItemText 313 method.

Agr – graphic object. This method returns object owned by RichViewEdit, do not not destroy it.

AVAlign – vertical alignment of the picture.

ATag – *tag* of the item. Use SetCurrentTag 349 or SetCurrentPictureInfo 348 to change *tag* of item as an editing operation. This value can also be read using GetCurrentTag 316 method.

This method must be called only when the document is formatted.

Unicode note: names of non-text items are ANSI strings.

Additional properties of item at the position of caret are returned by the methods GetCurrentItemExtraIntProperty

312 and GetCurrentItemExtraStrProperty 313.

See also methods:

- SetCurrentPictureInfo 348 (changes properties of picture at the position of caret, as an editing operation);
- GetPictureInfo 201 (returns properties of the specified picture).

See also properties:

- CurItemNo 288;
- CurItemStyle 289.

See also:

- Obtaining RichView items 50;
- Item types 29;
- "Tags" 34.

1.4.1.2.48 TRichView Edit.GetCurrentTag

Returns tag of the item at the caret position

```
function GetCurrentTag: TRVTag 645;
```

The same as TopLevelEditor 295.**GetItemTag** 195(TopLevelEditor 295.CurItemNo 288).

Use SetCurrentTag 349 to change tag of item at the caret position, as an editing operations.

This method must be called only when the document is formatted.

See also methods:

- SetCurrentTag 349.

See also properties:

- CurItemNo 288.

See also

- "Tags" 34.

1.4.1.2.49 TRichView Edit.GetCurrentTextInfo

Returns text and *tag* for text item at the position of caret

```
procedure GetCurrentTextInfo(var AText: String; var ATag: TRVTag 645);
```

GetCurrentTextInfo(...) is equivalent to TopLevelEditor 295.**GetTextInfo** 204(TopLevelEditor 295.CurItemNo 288, ...).

Use SetCurrentTag 349 to change tag of item at the caret position, as an editing operations.

This method must be called only when the document is formatted.

See also properties:

- CurItemNo 288;
- CurItemStyle 289.

See also:

- Obtaining items 50;
- Item types 29;
- "Tags" 34.

1.4.1.2.2.50 TRichView Edit.InsertBreak

Inserts *break* 93 (horizontal line) in the position of caret.

```
function InsertBreak(Width: Byte; Style: TRVBreakStyle624; Color: TColor):
Boolean;
```

Parameters:

Width – width of line (can be in range from 1 to 10);

Style – break type;

Color – color of line. If it is equal to *clNone*, the *break* has the color of the 0th text style 404.

Method type: editing-style method for insertion.

Return value:

True if the insertion was successful (it can fail due to protection, or when inserting in table having multicell selection)

See also methods of TCustomRichView:

- AddBreakEx and others 160.

See also:

- Inserting items in position of caret 52;
- Item types 29.

1.4.1.2.2.51 TRichView Edit.InsertBullet

Inserts *bullet* 96 in the position of caret.

```
function InsertBullet(ImageIndex: Integer; ImageList: TCustomImageList): Boolean;
```

Parameters:

Name – name of *bullet*, any string. **Name** must not contain CR and LF characters. RichViewEdit does not use item names itself, they are for your own use. This is an ANSI string, regardless of Unicode mode of text styles.

ImageIndex – index of image in **ImageList**.

ImageList – image list for this *bullet*. RichView does not own, does not copy and does not destroy image lists, it just holds pointers to them. So this image list is not destroyed when the document is cleared.

Method type: editing-style method for insertion.

Unicode note: names of non-text items are ANSI strings.

Return value:

True if the insertion was successful (it can fail due to protection, or when inserting in table having multicell selection)

See also methods of TCustomRichView:

- AddBulletEx and others 160;

See also:

- Inserting items in position of caret 52;
- Item types 29.

1.4.1.2.2.52 TRichView Edit.InsertCheckpoint

Inserts *checkpoint* in the position of caret.

```
procedure InsertCheckpoint(const ATag: TRVTag; const AName: String;
 ARaiseEvent: Boolean);
```

(introduced in version 10)

This method is similar to SetCurrentCheckpointInfo, but inserts *checkpoint* exactly in the position of caret.

If the caret is at the beginning of item, this methods calls SetCheckpointInfoEd to add *checkpoint* for this item.

If the caret is at the middle of text item, this method splits this item at the position of caret, and calls SetCheckpointInfoEd to add *checkpoint* for the item to the right.

If the caret is at the end of paragraph, the method moves it to the beginning of the next paragraph, and calls SetCheckpointInfoEd to add *checkpoint* for the first paragraph item (or the second item, if the first one is list marker).

If the caret is at the end of document, this methods does nothing.

This method is used in conjunction with GetCheckpointAtCaret and RemoveCheckpointAtCaret (GetCurrentCheckpoint and RemoveCurrentCheckpoint cannot be used, because *checkpoint* added by this method is not necessary belongs to the current item (that is usually the item to the left of the caret).

Parameters.

ATag – *tag* of *checkpoint*;

AName – name of the *checkpoint*, any string without line break (CR, LF) characters.

ARaiseEvent – "raise event" flag; if set, RichView can generate OnCheckpointVisible event for this *checkpoint*.

Method type: editing-style.

Unicode note: names of *checkpoints* are Unicode strings for Delphi/C++Builder 2009 or newer, and ANSI strings for older versions of Delphi/C++Builder.

See also:

- Modifying RichView items;
- "Checkpoints";
- "Tags".

1.4.1.2.2.53 TRichView Edit.InsertControl

Inserts Delphi/C++Builder control in the position of caret.

```
function InsertControl(const Name: TRVAnsiString; ctrl: TControl;
  VAlign: TRVVAlign): Boolean;
```

Parameters:

Name – name of this control item, any string. **Name** must not contain CR and LF characters. RichView does not use item names itself, they are for your own use. This is an ANSI string, regardless of Unicode mode of text styles. Do not confuse with **ctrl**.Name property.

ctrl – control to insert. You should create control but should not destroy it (RichViewEdit will do it for you).

VAlign – vertical align of this control, relative to its line, see TRVVAlign for possible values.

Method type: editing-style method for insertion.

Unicode note: names of non-text items are ANSI strings.

Return value

True if the insertion was successful (it can fail due to protection, or when inserting in table having multicell selection)

See also methods of TCustomRichView:

- AddControlEx and others 162;

See also:

- Inserting items in position of caret 52;
- Item types 29.

1.4.1.2.2.54 TRichView Edit.InsertHotPicture

Inserts picture with hypetext link 93 in the position of caret.

```
function InsertHotPicture(const Name: TRVAnsiString 623; gr: TGraphic;
  VAlign: TRVVAlign 648): Boolean;
```

(introduced in version 1.5)

The same as InsertPicture 322, but this picture will be a hypertext link.

Method type: editing-style method for insertion.

Return value:

True if the insertion was successful (it can fail due to protection, or when inserting in table having multicell selection). If insertion failed, TRichViewEdit destroys **gr**.

See also methods of TCustomRichView:

- AddHotPicture and others 163.

See also:

- Inserting items in position of caret 52;
- Hypertext in RichView 35;
- Item types 29.

1.4.1.2.2.55 TRichView Edit.InsertHotspot

Inserts *hotspot* 97 in the position of caret.

```
function InsertHotspot(ImageIndex, HotImageIndex: Integer;
  ImageList: TCustomImageList): Boolean;
```

Parameters:

Name – name of *hotspot*, any string. **Name** must not contain CR and LF characters. RichViewEdit does not use item names itself, they are for your own use. This is an ANSI string, regardless of Unicode mode of text styles.

ImageIndex – index of image in **ImageList**.

HotImageIndex – index of "hot" image in **ImageList**. It is displayed when the mouse pointer is above this *hotspot*.

ImageList – image list for this *bullet*. RichView does not own, does not copy and does not destroy image lists, it just holds pointers to them. So this image list is not destroyed when the document is cleared.

Method type: editing-style method for insertion.

Unicode note: names of non-text items are ANSI strings.

Return value:

True if the insertion was successful (it can fail due to protection, or when inserting in table having multicell selection)

See also methods of TCustomRichView:

- AddHotspotEx and others 164.

See also:

- Inserting items in position of caret 52;
- Hypertext in RichView 35;
- Item types 29.

1.4.1.2.2.56 TRichView Edit.InsertItem

Inserts item in the position of caret.

```
function InsertItem(const Name: String;
  Item: TCustomRVItemInfo 526): Boolean;
```

(introduced in version 1.4)

This is the most general method for inserting items. It can be used to insert items of custom types.

This method is usually used for inserting tables 98, labels 101, sequences 102, footnotes 104, endnotes 103, references to notes 108.

The detailed description of item types is beyond the scope of this help file. Briefly, standard item types are declared and implemented in RVItem unit. All item types are classes derived from one base class – TCustomRVItemInfo.

Parameters:

Text – some text associated with this item (the name of item). It must not contain CR and LF characters.

Item – item itself. **Item**.ParaNo defines paragraph style of item (index in Style.ParaStyles 398).

Method type: editing-style method for insertion.

Unicode note: for Delphi/Unicode 2009 or newer, Text is Unicode string; for the older versions of Delphi/C++Builder, Text is ANSI string. It is converted to ANSI/Unicode, if necessary. Names of non-text items are ANSI strings.

Return value:

If *True*, the item was inserted successfully.

If *False*, the item can not be inserted (because text is read-only 294 or protected 439, or when inserting in table having multicell selection); in this case **Item** was destroyed (freed) by InsertItem.

See also methods of TCustomRichView:

- AddItem 166.

See also:

- Inserting items in position of caret 52;
- Item types 29.

1.4.1.2.2.57 TRichView Edit.InsertOEMTextFromFile

Loads text file (in OEM-defined character set) **FileName** and inserts its content in the position of caret.

```
function InsertOEMTextFromFile(const FileName: String):Boolean;
```

"OEM" stands for "original equipment manufacturer". This is character set of MS DOS.

The inserted text (one or more text items) has current text and paragraph styles.

Method type: editing-style method for insertion.

Unicode note: File content must be in OEM charset. Text will be converted to ANSI or to Unicode, depending on Unicode 446 property of the current text style 290.

Return value:

"Was reading from from file successful?"

See also methods:

- InsertTextFromFile 328;
- InsertText 327.

See also properties:

- CurParaStyleNo 289;
- CurTextStyleNo 290.

See also properties of TRVStyle:

- TextStyles 404;
- ParaStyles 398;
- SpacesInTab 402.

See also:

- Saving and loading 58;
- Inserting items in position of caret 52.

1.4.1.2.2.58 TRichView Edit.InsertPageBreak

Inserts explicit page break in the position of caret.

```
procedure InsertPageBreak
```

(introduced in version 1.2)

The method deselects the document before performing its work.

If the caret is not at the beginning of some item which is always displayed from the new line (see IsFromNewLine 207), the method breaks the current line (just like if user pressed Enter key).

The method sets "explicit page break" flag for the item in the position of caret (see PageBreaksBeforeItems 149).

Method type: editing-style.

See also:

- RemoveCurrentPageBreak 334.

See also properties of RichView:

- PageBreaksBeforeItems 149;
- Options 146 (*rvoShowPageBreaks*).

See also properties of RVStyle:

- PageBreakColor 398.

1.4.1.2.2.59 TRichView Edit.InsertPicture

Inserts picture [90] in the position of caret.

```
function InsertPicture(const Name: TRVAnsiString[623]; gr: TGraphic;
  VAlign: TRVVAlign[648]): Boolean;
```

Parameters:

Name – name of this picture item, any string. **Name** must not contain CR and LF characters. RichView does not use items names itself, they are for your own use. This is an ANSI string, regardless of Unicode mode of text styles.

gr – picture to insert. By default, this picture will be owned by TRichView control, and you must not free it. However, you can specify nonzero *rvepShared* extra item property [627], and this graphic object will be shared (you need to free it yourself after TRichView is cleared).

VAlign – vertical align of this picture, relative to its line, see TRVVAlign [648] for possible values.

Method type: editing-style method for insertion.

Unicode note: names of non-text items are ANSI strings.

If you want this picture to be a hypertext link, use InsertHotPicture [319] instead.

Return value

True if the insertion was successful (it can fail due to protection, or when inserting in table having multicell selection). If insertion failed, TRichViewEdit destroys **gr**.

Example (inserting icon from file)

```
var ico: TIcon;
...
ico := TIcon.Create;
ico.LoadFromFile('My Icon.ico');
MyRichViewEdit.InsertPicture('',ico, rvvaBaseline);
```

Example 2 (inserting picture of any supported format):

```
var gr: TGraphic;
...
if OpenPictureDialog1.Execute then
  gr := RVGraphicHandler[662].LoadFromFile[613](OpenPictureDialog1.FileName);
  if gr<>nil then
    MyRichViewEdit.InsertPicture('', gr, rvvaBaseline)
  else
    // process a picture loading error here
end;
```

See also methods of TCustomRichView:

- AddPicture and others [169].

See also:

- Inserting items in position of caret [52];
- Hypertext in RichView [35];
- Item types [29].

1.4.1.2.2.60 TRichView Edit.InsertRTFFromFileEd

Inserts RTF (Rich Text Format) from file **FileName** in the position of caret.

```
function InsertRTFFromFileEd(const FileName: String): Boolean;
```

(Introduced in version 1.5)

Parameters for loading are in TRichView.RTFReadProperties[151]. The most important properties affecting RTF loading are RTFReadProperties.TextStyleMode[274] and ParaStyleMode[272]. The imported RTF may look absolutely different depending on values of these properties! Setting for RTF loading can be changed in the TRichView component editor[134].

Method type: editing-style method for insertion.

If style templates are used[158], and RTFReadProperties.TextStyleMode=RTFReadProperties.ParaStyleMode=*rvrsAddIfNeeded*, and StyleTemplateInsertMode[156]<>*rvstimIgnoreSourceStyleTemplates*, the method merges an RTF style sheet into Style[156].StyleTemplates[402], and reads text and paragraph styles according to StyleTemplateInsertMode[156]. The method calls OnStyleTemplatesChange[266] event.

Return value:

"successful reading?" (extended information is in TRichView.RTFReadProperties.ErrorCode[270])

Unicode note: Text can be imported as Unicode or not, depending on value of RTFReadProperties.UnicodeMode[274].

See also methods of TRichViewEdit:

- InsertRTFFromStreamEd[323];
- PasteRTF[330].

See also methods of TRichView:

- LoadRTF[209].

See also events of TRichView:

- OnImportPicture[242];
- OnReadHyperlink[248];
- OnStyleTemplatesChange[266],

See also:

- Saving and loading[58];
- Inserting items in position of caret[52].

1.4.1.2.2.61 TRichView Edit.InsertRTFFromStreamEd

Inserts RTF (Rich Text Format) from the **Stream** in the position of caret.

```
function InsertRTFFromStreamEd(Stream: TStream): Boolean;
```

(Introduced in version 1.5)

Parameters for loading are in TRichView.RTFReadProperties[151]. The most important properties affecting RTF loading are RTFReadProperties.TextStyleMode[274] and ParaStyleMode[272]. The imported RTF may look absolutely different depending on values of these properties! Setting for RTF loading can be changed in the TRichView component editor[134].

Method type: editing-style method for insertion.

If style templates are used[158], and RTFReadProperties.TextStyleMode=RTFReadProperties.ParaStyleMode=*rvrsAddIfNeeded*, and StyleTemplateInsertMode[156]<>*rvstimIgnoreSourceStyleTemplates*, the method merges an RTF style sheet into Style[156].StyleTemplates[402], and reads text and paragraph styles according to StyleTemplateInsertMode[156]. The method calls OnStyleTemplatesChange[266] event.

Return value:

"successful reading?" (extended information is in TRichView.RTFReadProperties.ErrorCode[270])

Unicode note: Text can be imported as Unicode or not, depending on value of RTFReadProperties.UnicodeMode[274].

See also methods of TRichViewEdit:

- InsertRTFFromFileEd[323];
- PasteRTF[330].

See also methods of TRichView:

- LoadRTFFromStream[210].

See also events of TRichView:

- OnImportPicture[242];
- OnReadHyperlink[248];
- OnStyleTemplatesChange[266].

See also:

- Saving and loading[58];
- Inserting items in position of caret[52].

1.4.1.2.2.62 TRichViewEdit.InsertRVFFromFileEd

Inserts RVF (RichView Format[59]) from the file **FileName** in the position of caret.

```
function InsertRVFFromFileEd(const FileName: String):Boolean;
```

Settings for RVF loading can be changed in the TRichView component editor[134].

Method type: editing-style method for insertion.

If style templates are used[158], and RVFTextStylesReadMode[154]=RVFParaStylesReadMode[154] =*rvf_sInsertMerge*, and StyleTemplateInsertMode[156]<>*rvstimIgnoreSourceStyleTemplates*, the method merges style templates from RVF into Style[156].StyleTemplates[402], and reads text and paragraph styles according to StyleTemplateInsertMode[156]. The method calls OnStyleTemplatesChange[266] event.

Return value: "successful reading?"

See also methods of TRichViewEdit:

- InsertRVFFromStreamEd[325];
- PasteRVF[331].

See also methods of TRichView:

- LoadRVF[211].

See also properties:

- RVFOptions[152];
- RVFWarnings[155];
- RVFTextStylesReadMode[154] (this method type: RVF insertion);
- RVFParaStylesReadMode[154] (this method type: RVF insertion);
- UseStyleTemplates[158];
- StyleTemplateInsertMode[156].

See also events:

- OnRVFImageListNeeded[252];
- OnRVFControlNeeded[251];
- OnRVFPictureNeeded[252];
- OnControlAction[239] (ControlAction=*rvcaAfterRVFLoad*).

See also:

- TRichView component editor 134;
- Saving and loading 58;
- Inserting items in position of caret 52;
- RVF 59.

1.4.1.2.2.63 TRichView Edit.InsertRVFFromStreamEd

Inserts RVF (RichView Format 59) from the **Stream** in the position of caret.

```
function InsertRVFFromStreamEd(Stream: TStream):Boolean;
```

Settings for RVF loading can be changed in the TRichView component editor 134.

Method type: editing-style method for insertion.

If style templates are used 158, and RVFTextStylesReadMode 154=RVFParaStylesReadMode 154 =*rvf_sInsertMerge*, and StyleTemplateInsertMode 156<>*rvstimIgnoreSourceStyleTemplates*, the method merges style templates from RVF into Style 156.StyleTemplates 402, and reads text and paragraph styles according to StyleTemplateInsertMode 156. The method calls OnStyleTemplatesChange 266 event.

Return value: "successful reading?"

See also methods of TRichViewEdit:

- InsertRVFFromFileEd 324;
- PasteRVF 331.

See also methods of TRichView:

- LoadRVFFromStream 210;
- InsertRVFFromStream 206;
- AppendRVFFromStream 173.
- **See also properties:**

- RVFOptions 152;
- RVFWarnings 155;
- RVFTextStylesReadMode 154 (this method type: RVF insertion);
- RVFParaStylesReadMode 154 (this method type: RVF insertion);
- UseStyleTemplates 158;
- StyleTemplateInsertMode 156.

See also events:

- OnRVFImageListNeeded 252;
- OnRVFControlNeeded 251;
- OnRVFPictureNeeded 252;
- OnControlAction 239 (ControlAction=*rvcaAfterRVFLoad*);
- OnStyleTemplatesChange 266.

See also:

- TRichView component editor 134;
- Saving and loading 58;
- Inserting items in position of caret 52;
- RVF 59.

1.4.1.2.2.64 TRichView Edit.InsertStringTag -A -W

The methods Insert text string **s** with associated **Tag** in the position of caret.

```
function InsertStringTag(const s: String; const Tag: TRVTag[645]): Boolean;
function InsertStringATag(const s: TRVAnsiString[623]; const Tag: TRVTag[645]): Boolean;
function InsertStringWTag(const s: TRVUnicodeString[647]; const Tag: TRVTag[645]): Boolean;
```

(introduced in version 1.6)

Inserted text (one text item) has the current text[290] and current paragraph[289] style.

Methods type: editing-style method for insertion.

Unlike InsertText[327]:

- **s** parameter must not contain CR, LF characters (#13 and #10) and page break character (#12);
- TAB characters are always replaced with spaces, even if SpacesInTab[402]=0.
- the inserted string is not merged with surrounding text items, even if it has the same text style and *tag*.

Do not overuse these methods. Use InsertText[327] when possible.

Unicode note:

- **InsertStringATag:** if the current text style is Unicode, **s** text will be converted to Unicode (conversion is based on charset of current text style).
- **InsertStringWTag:** if the current text style is ANSI, **s** text will be converted to ANSI (conversion is based on charset of current text style).
- **InsertStringTag** works like **InsertStringWTag** in Delphi/C++Builder 2009 or newer, and like **InsertStringATag** in the older versions of Delphi/C++Builder.

Return value:

True if the insertion was successful (it can fail due to protection, or when inserting in table having multicell selection)

See also methods:

- InsertText, -A, -W[327].

See also properties:

- CurParaStyleNo[289];
- CurTextStyleNo[290].

See also properties of TRVStyle:

- TextStyles[404];
- ParaStyles[398];
- SpacesInTab[402].

See also:

- Inserting items in position of caret[52];
- Tags[34].

1.4.1.2.2.65 TRichView Edit.InsertTab

Inserts tabulator[89] in the position of caret.

```
function InsertTab: Boolean;
```

(introduced in v1.9)

Inserted tabulator has the current text[290] and the current paragraph[289] style.

This method always inserts a tabulator, regardless of RVStyle.SpacesInTab[402] mode.

If RVStyle.SpacesInTab[402]=0, this method does the same work as InsertText[327](#9).

Method type: editing-style method for insertion.

Return value:

True if the insertion was successful (it can fail due to protection, or when inserting in table having multicell selection)

See also method of TRichView:

- AddTab 170.

1.4.1.2.2.66 TRichView Edit.InsertText -A -W

The methods insert **text** in the position of caret.

```
procedure InsertText(const text: String; CaretBefore: Boolean=False);
procedure InsertTextA(const text: TRVAnsiString 623; CaretBefore: Boolean=False);
procedure InsertTextW(const text: TRVUnicodeString 647; CaretBefore: Boolean=False);
```

Inserted text has current text 290 and current paragraph 289 style.

Text may contain special characters:CR, LF, TAB, FF (#13, #10, #9, #12). #9 characters may be inserted as tabulators 89, depending on value of SpacesInTab 402 property of the linked 156 RVStyle component. #12 characters add page breaks 149. All possible line breaks modes (CR, LF, CR+LF, LF+CR) are supported.

InsertTextW supports the Unicode byte order marks characters (if it's present, it must be the first character in **text**).

If **CaretBefore**=*True*, the caret will be positioned before the inserted text after insertion (default positioning is after the text). Set it to *True* when performing search-and-replace in down-up direction (note: the standard replace dialog does not allow to choose down-up direction) to avoid infinite loop (when replacing, for example, 'a' with 'aa'). The "smart start" search option helps to solve this problem; however, this advice is still in effect.

Methods type: editing-style method for insertion.

Unicode note:

- **InsertTextA:** if the current text style is Unicode, **text** will be converted to Unicode (conversion is based on the charset of the current text style).
- **InsertTextW:** if the current text style is ANSI, **text** will be converted to ANSI (conversion is based on the charset of the current text style).
- **InsertText** works like **InsertTextW** in Delphi/C++Builder 2009+, and like **InsertTextA** in older versions of Delphi/C++Builder.

See also methods:

- InsertStringTag, InsertStringATag, InsertStringWTag 326.

See also properties:

- CurParaStyleNo 289;
- CurTextStyleNo 290.

See also properties of TRVStyle:

- TextStyles 404;
- ParaStyles 398;
- SpacesInTab 402.

See also:

- Inserting items in position of caret 52.

1.4.1.2.2.67 TRichView Edit.InsertTextFromFile -W

The methods load text file **FileName** and insert its content in the position of caret. **InsertTextFromFile** inserts text from ANSI text files, **InsertTextFromFileW** inserts text from Unicode (UTF-16) text files.

```
function InsertTextFromFile(const FileName: String; CodePage: Cardinal=CP_ACP): Boolean;
function InsertTextFromFileW(const FileName: String): Boolean;
```

Inserted text has current text 290 and current paragraph 289 style.

A file may contain special characters:CR, LF, TAB, FF (#13, #10, #9, #12). #9 characters may be inserted as tabulators 89, depending on value of SpacesInTab 402 property of the linked 156 RVStyle component. #12 characters add page breaks 149. All possible line breaks modes (CR, LF, CR+LF, LF+CR) are supported.

InsertTextFromFileW supports the Unicode byte order marks characters (if they are present, they must be the first character in file).

Methods type: editing-style method for insertion.

Unicode note:

- **InsertTextFromFile:** if the current text style is Unicode, text will be converted to Unicode (conversion is performed using the **CodePage** parameter, or, of it's equal to CP_ACP, basing on the charset of the current text style). Note that if the current text style is not a Unicode style, **CodePage** is ignored.
- **InsertTextFromFileW:** if the current text style is ANSI, text will be converted to ANSI (conversion is based on the charset of the current text style).

Return value: "Was reading from from the file successful?"

See also methods:

- InsertOEMTextFromFile 321;
- InsertText, -A, -W 327.

See also methods of TRichView:

- LoadText, LoadTextW 212.

See also properties:

- CurParaStyleNo 289;
- CurTextStyleNo 290.

See also properties of TRVStyle:

- TextStyles 404;
- ParaStyles 398;
- SpacesInTab 402.

See also:

- Saving and loading 58;
- Inserting items in position of caret 52.

1.4.1.2.2.68 TRichView Edit.Paste

Pastes document from the Clipboard.

```
procedure Paste;
```

First, it generates OnPaste 365 event, allowing the programmer to insert data in custom format.

If DoDefault parameter of OnPaste was set to *False*, Paste finishes.

The method takes AcceptPasteFormats 287 property into account: it tries to paste the specific format only if it is included in the property.

Next, it tries to paste as RVF, see PasteRVF 331.

If unsuccessful, it tries to paste as RTF, see PasteRTF 330.

If unsuccessful, it tries to paste as an URL, see PasteURL 332.

If unsuccessful, it tries to paste as text, see PasteTextA and PasteTextW 331 (if the current text style is ANSI, it uses PasteTextA; if Unicode – PasteTextW).

If unsuccessful, it tries to paste image files, see PasteGraphicFile 329.

If unsuccessful, it tries to paste as a bitmap, see PasteBitmap 329.

If unsuccessful, it tries to paste as a metafile picture, see PasteMetafile 330.

This method is executed automatically when user presses Ctrl+V or Shift+Insert.

Method type: editing-style method for insertion.

See also methods:

- CanPaste 303.

See also:

- RichView and Clipboard 48.

1.4.1.2.2.69 TRichView Edit.PasteBitmap

Inserts Windows bitmap from the Clipboard in the position of caret

```
function PasteBitmap(TextAsName: Boolean): Boolean;
```

This method does nothing if the Clipboard does not contain Windows bitmap;

It inserts a new picture item with bitmap (TBitmap) from the Clipboard; if **TextAsName**=*True* and the Clipboard contains text, it sets name of this item to the text from the Clipboard (warning: if you want to save this document to RVF, names of items should not contain CR and LF characters).

Method type: editing-style method for insertion.

Unicode note: names of non-text items are ANSI strings.

Return value:

True, if image was inserted.

See also methods:

- Paste 328;
- InsertPicture 322.

See also properties:

- DefaultPictureVAlign 291.

See also:

- RichView and Clipboard 48.

1.4.1.2.2.70 TRichView Edit.PasteGraphicFiles

Inserts graphic files from the Clipboard (CF_HDROP format).

```
function TCustomRichViewEdit.PasteGraphicFiles(
  FileNamesAsNames,StoreFileNames: Boolean): Boolean;
```

(introduced in version 10)

This method does nothing if the Clipboard does not contain graphic files in CF_HDROP format. This format is copied, for example, by Windows Explorer, or by Internet Explorer, when user copies an image (a link to the file in the browser's cache is copied).

Parameters:

If **FileNamesAsNames** = *True*, the full file name is stored in the picture item name.

If **StoreFileNames** = *True*, the full file name is is stored in *rvespImageFileName* extra string property 630.

Method type: editing-style method for insertion.

Unicode note: names of non-text items and extra item string properties are ANSI strings.

Return value:

True, if at least one of pictures was inserted.

See also methods:
- Paste 328;
- InsertPicture 322.

See also properties:
- DefaultPictureVAlign 291.

See also:
- RichView and Clipboard 48.

1.4.1.2.71 TRichView Edit.PasteMetafile

Inserts metafile from the Clipboard in the position of caret

```
function PasteMetafile(TextAsName: Boolean): Boolean;
```

This method does nothing if the Clipboard does not contain a Windows metafile.

It inserts a new picture item with metafile (TMetafile) from the Clipboard; if **TextAsName**=*True* and the Clipboard contains text, it sets name of this item to the text from the Clipboard (warning: if you want to save this document to RVF, names of items should not contain CR and LF characters).

Method type: editing-style method for insertion.

Unicode note: names of non-text items are ANSI strings.

Return value:

True, if image was inserted.

See also methods:
- Paste 328;
- InsertPicture 322.

See also properties:
- DefaultPictureVAlign 291.

See also:
- RichView and Clipboard 48.

1.4.1.2.72 TRichView Edit.PasteRTF

Inserts RTF (Rich Text Format) from the Clipboard in the position of caret

```
procedure PasteRTF;
```

(Introduced in version 1.5)

This method does nothing if the Clipboard does not contain RTF.

If the Clipboard has RTF, and there was an error during reading, you can get additional information in TRichView.RTFReadProperties.ErrorCode 270 property.

Parameters for loading are in TRichView.RTFReadProperties 151. The most important properties affecting RTF loading are RTFReadProperties.TextStyleMode 274 and ParaStyleMode 272. The imported RTF may look absolutely different depending on values of these properties! Setting for RTF loading can be changed in the

TRichView component editor 134.

If the Clipboard contains HTML format as well, RichViewEdit reads base path for external images from it.

Method type: editing-style method for insertion.

If style templates are used 158, and RTFReadProperties.TextStyleMode=RTFReadProperties.ParaStyleMode=*rvrsAddIfNeeded*, and StyleTemplateInsertMode 156<>*rvstimIgnoreSourceStyleTemplates*, the method merges an RTF style sheet into Style 156.StyleTemplates 402, and reads text and paragraph styles according to StyleTemplateInsertMode 156. The method calls OnStyleTemplatesChange 266 event.

Unicode note: Text can be imported as Unicode or not, depending on value of RTFReadProperties.UnicodeMode 274.

See also methods:

- CanPasteRTF 303;
- InsertRTFFromFileEd 323;
- InsertRTFFromStreamEd 323.

See also:

- RichView and Clipboard 48

1.4.1.2.2.73 TRichView Edit.PasteRVF

Inserts RVF (RichView Format) from the Clipboard in the position of caret

```
procedure PasteRVF;
```

This method does nothing if the Clipboard does not contain RVF.

Method type: editing-style method for insertion.

See also methods:

- CanPasteRVF 303;
- Paste 328;
- InsertRVFFromStreamEd 325.

See also:

- RichView and Clipboard 48;
- RVF 59.

1.4.1.2.2.74 TRichView Edit.PasteText -A -W

These methods Insert text from the Clipboard in the position of caret

```
procedure PasteText;
procedure PasteTextA;
procedure PasteTextW;
```

PasteTextA pastes text in CF_TEXT format. **PasteTextW** pastes text in CF_UNICODETEXT.

In Windows 95/98/ME, CF_TEXT and CF_UNICODETEXT are independent Clipboard formats.

In Windows NT/2000/XP/Vista, Windows automatically converts them to each other, using the language of the keyboard layout that was used at the moment of copying.

These methods do nothing if the Clipboard does not contain ANSI/Unicode text.

Text is inserted in the position of caret, using the current text 290 and paragraph 289 styles.

Text from the Clipboard may contain special characters: CR, LF, TAB, FF (#13, #10, #9, #12). #9 characters may be inserted as tabulators 89, depending on value of SpacesInTab 402 property of the linked 156 RVStyle component. #12 characters add page breaks 149. All possible line breaks modes (CR, LF, CR+LF, LF+CR) are supported. **PasteTextW** supports the Unicode byte order marks characters (if it's present, it must be the first

character).

Methods type: editing-style method for insertion.

Unicode note:

- **PasteTextA:** if the current text style is Unicode, text from the Clipboard will be converted to Unicode (conversion is based on the charset of the current text style).
- **PasteTextW:** if the current text style is ANSI, text from the Clipboard will be converted to ANSI (conversion is based on the charset of the current text style).
- **PasteText** works like PasteTextW in Delphi/C++Builder 2009+, and like PasteTextA in the older versions of Delphi/C++Builder.

See also methods:

- Paste 328;
- InsertText, -A, -W 327.

See also properties:

- CurParaStyleNo 289;
- CurTextStyleNo 290.

See also properties of TRVStyle:

- TextStyles 404;
- ParaStyles 398;
- SpacesInTab 402.

See also:

- RichView and Clipboard 48.

1.4.1.2.75 TRichView Edit.PasteURL

Inserts URL from the Clipboard in the position of caret

```
function PasteURL: Boolean;
```

(introduced in version 14)

This method does nothing and returns *False* if the Clipboard does not contain URL.

The method calls OnReadHyperlink 248 event, passing the current text style in the StyleNo parameter. The application should return the corresponding hypertext style in this parameter (if the event is not processed, URL is inserted as a plain text with URL in its tag 34).

Method type: editing-style method for insertion.

See also methods:

- Paste 328.

See also:

- RichView and Clipboard 48.

1.4.1.2.76 TRichView Edit.Redo

Redoes the last undone editing operation.

```
procedure Redo;
```

(Introduced in version 1.3)

This method does nothing if there is no operation to redo.

Information about the operation to redo is returned by the methods RedoAction 333 and RedoName 333.

This method must be called only when the document is formatted.

This method is called when the user presses Shift+Ctrl+Z or Shift+Alt+Backspace.

Method type: editing-style.

See also:
- Undo 355;
- Undo in RichViewEdit 54.

1.4.1.2.2.77 TRichView Edit.RedoAction

Returns type of the operation to be redone 332 next.

```
function RedoAction: TRVUndoType 647;
```

(Introduced in version 1.3)

See TRVUndoType 647 for the list of operations. If the returned value is *rvutCustom*, you can use RedoName 333.

See also:
- UndoAction 355;
- Undo in RichViewEdit 54.

1.4.1.2.2.78 TRichView Edit.RedoName

Returns name of the operation to be redone 332 next.

```
function RedoName: String;
```

(Introduced in version 1.3)

Returns empty string for the standard undo/redo operations, i.e. all operations but *rvutCustom* (see RedoAction 333).

Returns non-empty string only for custom redo action, initiated by BeginUndoCustomGroup 302.

See also:
- UndoName 355;
- Undo in RichViewEdit 54.

1.4.1.2.2.79 TRichView Edit.RemoveCheckpointAtCaret

Deletes *checkpoint* at the position of caret.

```
procedure RemoveCheckpointAtCaret;
```

(introduced in version 10)

This method deletes *checkpoint* only if the caret is at the beginning of item.

This method deletes the *checkpoint* that is returned by GetCheckpointAtCaret 307.

This method is used in conjunction with InsertCheckpoint 318 and GetCheckpointAtCaret 307. The alternative set of methods is: SetCurrentCheckpointInfo 343, GetCurrentCheckpoint 309 and RemoveCurrentCheckpoint 334.

Method type: editing-style, unlike RemoveCheckpoint 214.

See also:
- "Checkpoints" 30.

1.4.1.2.2.80 TRichView Edit.RemoveCheckpointEd

Editing-style method for deleting *checkpoint* associated with the **ItemNo**-th item.

```
procedure RemoveCheckpointEd(ItemNo: Integer);
```

Method type: editing-style, unlike RemoveCheckpoint 214.

See also methods of TRichViewEdit:

- RemoveCurrentCheckpoint 334;
- SetCheckpointInfoEd 340.

See also methods of TRichView:

- GetItemCheckpoint 191;
- RemoveCheckpoint 214.

See also:

- Modifying RichView items 51;
- "Checkpoints" 30.

1.4.1.2.2.81 TRichView Edit.RemoveCurrentCheckpoint

Deletes *checkpoint* associated with the item at the position of caret.

```
procedure RemoveCurrentCheckpoint;
```

RemoveCurrentCheckpoint is equivalent to TopLevelEditor 295.RemoveCheckpointEd 333(TopLevelEditor 295.CurItemNo 288).

Method type: editing-style.

See also methods:

- GetCurrentCheckpoint 309;
- SetCurrentCheckpointInfo 343;
- RemoveCheckpointEd 333.

See also properties:

- CurItemNo 288.

See also:

- Modifying RichView items 51;
- "Checkpoints" 30.

1.4.1.2.2.82 TRichView Edit.RemoveCurrentPageBreak

Removes an explicit page break before the paragraph containing the caret.

```
procedure RemoveCurrentPageBreak;
```

(Introduced in version 1.2)

The method removes "explicit page break" flag of the first item belonging to the paragraph containing the caret (see PageBreaksBeforeItems 149), if this flag was set.

Method type: editing-style.

See also:

- InsertPageBreak 321.

See also properties of RichView:

- PageBreaksBeforeItems 149;
- Options 146 (*rvoShowPageBreaks*).

See also properties of RVStyle:

- PageBreakColor 398.

1.4.1.2.2.83 TRichView Edit.RemoveLists

Removes markers 99 (bullets and numbering) from the selected paragraphs.

```
procedure RemoveLists(ARecursive: Boolean);
```

(introduced in version 1.7)

Parameter:

ARecursive – reserved for future use. Set it to *False*.

Method type: editing-style.

See also:
- ApplyListStyle 297;
- ChangeListLevels 304.

See also methods of TCustomRichView:
- SetListMarkerInfo 233;
- GetListMarkerInfo 199.

1.4.1.2.2.84 TRichView Edit.ResizeControl

Resizes control 94 in the **ItemNo**-th item, and quickly reformats the affected part of the document.

```
procedure ResizeControl(ItemNo, NewWidth, NewHeight: Integer);
```

The **ItemNo**-th item must be of "inserted control" 94 type (*rvsComponent* 663), otherwise the method raises ERichViewError 608 exception.

The control's width will be set to **NewWidth**, height to **NewHeight**.

Method type: editing-style.

See also methods:
- ResizeCurrentControl 335;
- SetControlInfoEd 341;
- AdjustControlPlacement 296;
- GetItemStyle 194.

See also properties:
- ItemCount 144.

See also:
- Modifying RichView items 51;
- Item types 29.

1.4.1.2.2.85 TRichView Edit.ResizeCurrentControl

Resizes control 94 at the position of caret and quickly reformats the affected part of the document.

```
procedure ResizeCurrentControl(NewWidth, NewHeight: Integer);
```

(introduced in version 1.4)

ResizeCurrentControl(...) is equivalent to TopLevelEditor 295.ResizeControl 335(TopLevelEditor 295.CurItemNo 288, ...).

Method type: editing-style.

See also methods:
- ResizeControl 335;
- SetControlInfoEd 341;
- AdjustControlPlacement 296;
- GetItemStyle 194.

See also properties:

- ItemCount 144.

See also:

- Modifying RichView items 51;
- Item types 29.

1.4.1.2.2.86 TRichView Edit.SearchText -W

The methods search for the substring **s** in the document.

```
function SearchText(s: String; SrchOptions: TRVESearchOptions 626): Boolean;
function SearchTextA(s: TRVAnsiString 623; SrchOptions: TRVESearchOptions 626): Boolean;
function SearchTextW(s: TRVUnicodeString 647; SrchOptions: TRVESearchOptions 626): Boolean;
```

When found, these methods select substring and move the caret to the end of selection.

If there is no selected fragment, the methods start searching from the caret position, up or down depending on **SrchOptions**.

If something is selected, *rvseoSmartStart* is used. If it is included, then:

- when searching down, the search starts from the second selected character;
- when searching up, the search starts from the last but one selected character;

(selected characters are counted from the top of document, the selection direction is ignored). This option allows to avoid selecting the same fragment again when changing a search direction.

If *rvseoMultiItem* is included, the search can match substrings of several text items 88. If not included, the text is searched in each text item separately. For example, if the document contains the text **He*llo***, the substring 'Hello' can be found only if this option is included, because 'He' and 'llo' belong to different items.

Unicode note: The methods allow searching both in ANSI and in Unicode 64 text items. When **SearchTextA** compares Unicode text with **s**, the conversion is based on RVStyle.DefCodePage 388. When **SearchTextW** compares ANSI text with **s**, the conversion is based on the character set of the text style. **SearchText** works like **SearchTextW** in Delphi/C++Builder 2009 or newer, and like **SearchTextA** in the older versions.

These methods can be used for search-and-replace operation as well. **SearchText** selects the found string, InsertText 327 replaces it with the new one. See important note about search and replace in down-up direction in the topic about InsertText 327.

History and compatibility

Since version 1.2 SearchText can search upwards.
Since version 1.5 SearchText can search whole words (substring must be delimited with characters from Delimiters 140)
SearchTextW is introduced in version 10.
rvsroMultiItem is introduced in version 12.

Note: There is incompatibility with programs written with the older version of RichView (1.0 – 1.1.4). Now if rvsroDown is not included in SrchOptions, the methods search upwards. In the older versions there was no rvsroDown option, and SearchText searched downwards only.

These methods must be called only when the document is formatted.

Return value:

Is the substring found?

See also:

- TRichView.SearchText, -A, W 222;

- GetRVESearchOptions 616 global function;
- Searching in RichView and RichViewEdit 49.

1.4.1.2.2.87 TRichView Edit.SelectCurrentLine

Selects the line containing the caret.

```
procedure SelectCurrentLine;
```

(introduced in version 1.6)

The line beginning and ending position may depend on word wrapping.

This method repaints document, moves caret to the end of line and generates OnSelect 265 event.

This method must be called only when the document is formatted.

See also methods:

- SelectCurrentWord 337;
- GetSelText 204.

See also:

- Selecting in RichView document 47.

1.4.1.2.2.88 TRichView Edit.SelectCurrentWord

Selects word at the position of caret. If there is non text item at the caret position, this method does nothing.

```
procedure SelectCurrentWord;
```

This method repaints document, moves the caret to the end of word and generates OnSelect 265 event.

"Word" is defined as a part of string between delimiters. Delimiters are listed in Delimiters 140 property.

This method must be called only when the document is formatted.

See also methods:

- SelectWordAt 224;
- GetSelText 204;
- SelectCurrentLine 337.

See also:

- Selecting in RichView document 47.

1.4.1.2.2.89 TRichView Edit.Set*PropertyEd

Editing-style methods for changing properties of the editor.

```
procedure SetIntPropertyEd(Prop: TRVIntProperty 635; Value: Integer;
  AutoReformat: Boolean);
procedure SetStrPropertyEd(Prop: TRVStrProperty 643; const Value: String;
  AutoReformat: Boolean);
procedure SetFloatPropertyEd(Prop: TRVFloatProperty 632; Value: TRVLength 636;
  AutoReformat: Boolean);
```

(introduced in version 15)

Parameters:

Prop identifies the property to change. If the property is a property of enumerated type, use Ord(). If the property is a boolean property, any non-zero value means *True*.

Value – new property value.

If **AutoReformat**=*True*, the editor is reformatted (if necessary), and Change 303 is called. Normally, this method is called with **AutoReformat**=*True* if you change a single property. If you call these methods multiple times for

changing multiple properties, reformat (if necessary) and call Change yourself.

Method type: editing-style (unlike direct assignment to these properties).

Example 1: calling for a property of an enumerated type (using Ord):

```
MyRichViewEdit.SetIntPropertyEd(rvipBackgroundStyle, Ord(bsTiled), True);
```

Example 2: calling for a property of a boolean type (any nonzero value means *True*):

```
MyRichViewEdit.SetIntPropertyEd(rvipDPMirrorMargins, 1, True);
```

Example 3: calling for multiple properties:

```
//var NewMargins: TRect
if MyRichViewEdit.CanChange then begin
  MyRichViewEdit.BeginUndoGroup(rvutLayout);
  MyRichViewEdit.SetUndoGroupMode(True);
  MyRichViewEdit.SetIntPropertyEd(rvipLeftMargin,   NewMargins.Left,   False);
  MyRichViewEdit.SetIntPropertyEd(rvipTopMargin,    NewMargins.Top,    False);
  MyRichViewEdit.SetIntPropertyEd(rvipRightMargin,  NewMargins.Right,  False);
  MyRichViewEdit.SetIntPropertyEd(rvipBottomMargin, NewMargins.Bottom, False);
  MyRichViewEdit.SetUndoGroupMode(False);
  MyRichViewEdit.Reformat;
  MyRichViewEdit.Change;
end;
```

See also methods:

- SetBackgroundImageEd.

1.4.1.2.2.90 TRichView Edit.SetBackgroundImageEd

An editing-style method for changing BackgroundBitmap.

```
procedure SetBackgroundImageEd(Graphic: TGraphic; AutoReformat: Boolean);
```

(introduced in version 15)

Parameters:

Graphic – a graphic to assign to BackgroundBitmap. If **Graphic** is not TBitmap, it is converted to TBitmap. After calling this method, the component uses a copy of **Graphic**, so you must free **Graphic** yourself.

If **AutoReformat**=*True*, Change is called. Normally, this method is called with **AutoReformat**=*True* if you change a single property. If you call this method together with other methods in a group, this parameter must be *False*, and you must call Change yourself.

Method type: editing-style (unlike direct assignment to BackgroundBitmap).

Example:

```
if MyRichViewEdit.CanChange then begin
  MyRichViewEdit.BeginUndoGroup(rvutBackground);
  MyRichViewEdit.SetUndoGroupMode(True);
  MyRichViewEdit.SetIntPropertyEd(rvipBackgroundStyle, Ord(bsTiled), False);
  MyRichViewEdit.SetBackgroundImageEd(Image1.Picture.Graphic, False);
  MyRichViewEdit.SetUndoGroupMode(False);
  MyRichViewEdit.Invalidate;
  MyRichViewEdit.Change;
end;
```

See also methods:

- SetIntPropertyEd, SetStrPropertyEd, SetFloatPropertyEd.

1.4.1.2.2.91 TRichView Edit.SetBreakInfoEd

An editing-style method for changing properties of the **ItemNo**-th item of RichView, if this item is a *break* [93].

```
procedure SetBreakInfoEd(ItemNo: Integer; AWidth: TRVStyleLength[644];
  AStyle: TRVBreakStyle[624]; AColor: TColor; const ATag: TRVTag[645]);
```

Parameters:

ItemNo – index of the item. The item must be of *break* [93] type (*rvsBreak* [663]), otherwise the method raises ERichViewError [608] exception. Items are indexed from 0 to ItemCount [144]-1, GetItemStyle [194] returns type of item.

Items of subdocuments (table cells [567]) are not included in the items range of the main document. For items in cells, activate inplace-editor, select the proper item in the cell (SelSelectionBounds [235]), and call SetBreakInfoEd of this inplace-editor. Alternatively, you can use SetCurrentBreakInfo [342].

AWidth – line width (or rectangle height). This value is measured in Style [156].Units [404].

AStyle – visual style of this *break*, see TRVBreakStyle [624] for possible values.

AColor – line color. If it is equal to *clNone*, Style [156].TextStyles [404][0].Color [435] is used.

ATag – *tag* of the item. You can use value returned by GetBreakInfo [183] or GetItemTag [195] for this item.

Instead of this method, you can use SetItemExtraIntPropertyExEd [350] and SetItemTagEd [352] methods.

Method type: editing-style (unlike SetBreakInfo [226]).

Additional item properties are assigned by the methods SetItemExtraIntPropertyEd [350] and SetItemExtraStrPropertyEd [351].

When possible, use SetCurrentBreakInfo [342] instead of this method.

See also methods of TCustomRichView:

- GetBreakInfo [183];
- GetItemStyle [194];
- SetBreakInfo [226].

See also methods:

- SetCurrentBreakInfo [342].

See also properties:

- ItemCount [144].

See also:

- Modifying RichView items [51];
- Item types [29];
- "Tags" [34].

1.4.1.2.2.92 TRichView Edit.SetBulletInfoEd

An editing-style method for changing properties of the **ItemNo**-th item of RichView, if this item is a *bullet* [96] *or* *hotspot* [97].

```
procedure SetBulletInfoEd(ItemNo: Integer; const AName: TRVAnsiString[623];
  AImageIndex: Integer; AImageList: TCustomImageList; const ATag: TRVTag[645]);
```

Parameters:

ItemNo – index of the item. The item must be of *bullet* [96] or *hotspot* [97] type (*rvsBullet or rvsHotspot* [663]), otherwise the method raises ERichViewError [608] exception. Items are indexed from 0 to ItemCount [144]-1, GetItemStyle [194] returns type of item.

Items of subdocuments (table cells 567) are not included in the items range of the main document. For items in cells, activate inplace-editor, select the proper item in the cell (SelSelectionBounds 235), and call SetBulletInfoEd of this inplace-editor. Alternatively, you can use SetCurrentBulletInfo 342.

AName – name of *bullet* 96, any string without line break (CR, LF) characters. It can also be set using SetItemTextEd 352 method.

AImageList – not used, reserved, set it to *nil*.

AImageIndex – index of image in image list. It can also be set using SetItemExtraIntPropertyExEd 350 method.

ATag – *tag* of the item. You can use value returned by GetBulletInfo 184 or GetItemTag 195 for this item. The *tag* can also be set by SetItemTagEd 352 method.

Method type: editing-style (unlike SetBulletInfo 227).

Additional item properties are assigned by the methods SetItemExtraIntPropertyEd 350 and SetItemExtraStrPropertyEd 351.

Unicode note: names of non-text items are ANSI strings.

When possible, use SetCurrentBulletInfo 342 instead of this method.

See also methods of TCustomRichView:

- GetBulletInfo 184;
- GetItemStyle 194;
- SetBulletInfo 227.

See also methods:

- SetCurrentBulletInfo 342.

See also properties:

- ItemCount 144.

See also:

- Modifying RichView items 51;
- Item types 29;
- "Tags" 34.

1.4.1.2.2.93 TRichView Edit.SetCheckpointInfoEd

An editing-style method for creating new or modifying the existing *checkpoint* associated with the **ItemNo**-th item

```
procedure SetCheckpointInfoEd(ItemNo: Integer; const ATag: TRVTag 645;
  const AName: String; ARaiseEvent: Boolean);
```

Parameters

ItemNo – index of the item, from 0 to ItemCount 144-1. Items of subdocuments (table cells 567) are not included in the items range of the main document. For items in cells, activate inplace-editor, select the proper item in the cell (SelSelectionBounds 235), and call SetCheckpointInfoEd of this inplace-editor. Alternatively, you can use SetCurrentCheckpointInfo 343.

ATag – *tag* of *checkpoint*;

AName – name of the *checkpoint*, any string without line break (CR, LF) characters.

ARaiseEvent – "raise event" flag; if set, RichView can generate OnCheckpointVisible 238 event for this *checkpoint*.

Method type: editing-style (unlike SetCheckpointInfo 227).

Unicode note: names of *checkpoints* are Unicode strings for Delphi/C++Builder 2009 or newer, and ANSI strings for older versions of Delphi/C++Builder.

When possible, use SetCurrentCheckpointInfo[343] instead of this method.

See also methods of TCustomRichView:

- GetItemCheckpoint[191];
- SetCheckpointInfo[227].

See also methods:

- SetCurrentCheckpointInfo[343].

See also properties:

- ItemCount[144].

See also:

- Modifying RichView items[51];
- "Checkpoints"[30];
- "Tags"[34].

1.4.1.2.2.94 TRichView Edit.SetControlInfoEd

An editing-style method for changing properties of the **ItemNo**-th item of RichView, if this item is "inserted control"[94].

```
procedure SetControlInfoEd(ItemNo: Integer;
  const AName: TRVAnsiString[623]; AVAlign: TRVVAlign[648];
  const ATag: TRVTag[645]);
```

Parameters:

ItemNo – index of the item. The item must be of control[94] type (*rvsComponent*[663]), otherwise the method raises ERichViewError[608] exception. Items are indexed from 0 to ItemCount[144]-1, GetItemStyle[194] returns type of item.

Items of subdocuments (table cells[567]) are not included in the items range of the main document. For items in cells, activate inplace-editor, select the proper item in the cell (SelSelectionBounds[235]), and call SetControlInfoEd of this inplace-editor. Alternatively, you can use SetCurrentControlInfo[344].

AName – name of the item, any string without line break (CR, LF) characters. It can also be set using SetItemTextEd[352] method. This is not a Name property of the control!

AVAlign – vertical alignment of this item, see TRVVAlign[648] for possible options.

ATag – *tag* of the item. Do not confuse with Tag property of the control! You can use value returned by GetControlInfo[187] or GetItemTag[195] for this item. The *tag* can also be set by SetItemTagEd[352] method.

Method type: editing-style (unlike SetControlInfo[228]).

Additional item properties are assigned by the methods SetItemExtraIntPropertyEd[350] and SetItemExtraStrPropertyEd[351].

Unicode note: names of non-text items are ANSI strings.

When possible, use SetCurrentControlInfo[344] instead of this method.

See also methods of TCustomRichView:

- GetControlInfo[187];
- GetItemStyle[194];
- SetControlInfo[228].

See also methods:

- SetCurrentControlInfo 344;
- ResizeControl 335;
- AdjustControlPlacement 296.

See also properties:

- ItemCount 144.

See also:

- Modifying RichView items 51;
- Item types 29;
- "Tags" 34.

1.4.1.2.95 TRichView Edit.SetCurrentBreakInfo

Changes properties for the item at the position of caret, if this item is a *break* 93

```
procedure SetCurrentBreakInfo(AWidth: TRVStyleLength644;
  AStyle: TRVBreakStyle624;  AColor: TColor; const ATag: TRVTag645);
```

SetCurrentBreakInfo(...) is equivalent to TopLevelEditor 295.SetBreakInfoEd 339(TopLevelEditor 295.CurItemNo 288, ...).

This method can be used if CurItemStyle 289 returns *rvsBreak* 663.

Parameters:

AWidth – line width (or rectangle height). This value is measured in Style 156.Units 404.

AStyle – visual style of this *break*, see TRVBreakStyle 624 for possible values.

AColor – line color. If it is equal to *clNone*, Style 156.TextStyles 404[0].Color 435 is used.

ATag – *tag* of the item. You can use value returned by GetCurrentBreakInfo 308 or GetCurrentTag 316 for this item.

Instead of this method, you can use SetCurrentItemExtraIntPropertyEx 345 and SetCurrentTag 349 methods.

Method type: editing-style.

Additional item properties are assigned by the methods SetCurrentItemExtraIntProperty 345 and SetCurrentItemExtraStrProperty 346.

See also methods:

- GetCurrentBreakInfo 308;
- SetBreakInfoEd 339.

See also properties:

- CurItemNo 288;
- CurItemStyle 289.

See also:

- Modifying RichView items 51;
- Item types 29;
- "Tags" 34.

1.4.1.2.96 TRichView Edit.SetCurrentBulletInfo

Changes properties for the item at the position of caret, if this item is a *bullet* 96 or *hotspot* 97.

```
procedure SetCurrentBulletInfo(const AName: TRVAnsiString623;
  AImageIndex: Integer; AImageList: TCustomImageList;
  const ATag: TRVTag645);
```

SetCurrentBulletInfo(...) is equivalent to TopLevelEditor[295].**SetBulletInfoEd**[339](TopLevelEditor[295].CurItemNo[288], ...).

This method can be used if CurItemStyle[289] returns *rvsBullet or rvsHotspot*[663].

Parameters:

AName – name of *bullet*[96], any string without line break (CR, LF) characters. It can also be set using SetCurrentItemText[347] method.

AImageList – not used, reserved, set it to *nil*.

AImageIndex – index of image in image list. It can also be set using SetCurrentItemExtraIntPropertyEx[345] method.

ATag – *tag* of the item. You can use value returned by GetCurrentBulletInfo[308] or GetCurrentTag[316] for this item. The *tag* can also be set by SetCurrentTag[349] method.

Method type: editing-style.

Additional item properties are assigned by the methods SetCurrentItemExtraIntProperty[345] and SetCurrentItemExtraStrProperty[346].

Unicode note: names of non-text items are ANSI strings.

See also methods:

- GetCurrentBulletInfo[308];
- SetBulletInfoEd[339].

See also properties:

- CurItemNo[288];
- CurItemStyle[289].

See also:

- Modifying RichView items[51];
- Item types[29];
- "Tags"[34].

1.4.1.2.2.97 TRichView Edit.SetCurrentCheckpointInfo

Creates or modifies *checkpoint* associated with the item at the position of caret.

```
procedure SetCurrentCheckpointInfo(const ATag: TRVTag[645];
  const AName: String; ARaiseEvent: Boolean);
```

SetCurrentCheckpointInfo(...) is equivalent to TopLevelEditor[295].**SetCheckpointInfoEd**[340](TopLevelEditor[295].CurItemNo[288], ...).

If the item at the position of caret already has a *checkpoint* (see GetCurrentCheckpoint[309]), its properties are modified; otherwise, a new *checkpoint* is created.

This method is used in conjunction with GetCurrentCheckpoint[309] and RemoveCurrentCheckpoint[334]. The alternative set of methods is: InsertCheckpoint[318], GetCheckpointAtCaret[307] and RemoveCheckpointAtCaret[333].

Parameters.

ATag – *tag* of *checkpoint*;

AName – name of the *checkpoint*, any string without line break (CR, LF) characters.

ARaiseEvent – "raise event" flag; if set, RichView can generate OnCheckpointVisible[238] event for this *checkpoint*.

Method type: editing-style.

Unicode note: names of *checkpoints* are Unicode strings for Delphi/C++Builder 2009 or newer, and ANSI strings for older versions of Delphi/C++Builder.

See also methods:

- GetCurrentCheckpoint 309, RemoveCurrentCheckpoint 334;
- InsertCheckpoint 318, GetCheckpointAtCaret 307, RemoveCheckpointAtCaret 333;
- SetCheckpointInfoEd 340.

See also properties:

- CurItemNo 288.

See also:

- Modifying RichView items 51;
- "Checkpoints" 30;
- "Tags" 34.

1.4.1.2.2.98 TRichViewEdit.SetCurrentControlInfo

Modifies the item at the position of caret, if this item is an inserted control 94.

```
procedure SetCurrentControlInfo(const AName: TRVAnsiString 623;
  AVAlign: TRVVAlign 648; const ATag: TRVTag 645);
```

SetCurrentControlInfo(...) is equivalent to TopLevelEditor 295.**SetControlInfoEd** 341(TopLevelEditor 295.CurItemNo 288, ...).

This method can be used if CurItemStyle 289 returns *rvsComponent* 663.

Parameters:

AName – name of the item, any string without line break (CR, LF) characters. It can also be set using SetCurrentItemText 347 method. This is not a Name property of the control!

AVAlign – vertical alignment of this item, see TRVVAlign 648 for possible options.

ATag – *tag* of the item. Do not confuse with Tag property of the control! You can use value returned by GetCurrentControlInfo 309 or GetCurrentTag 316 for this item. The *tag* can also be set by SetCurrentTag 349 method.

Method type: editing-style.

Additional item properties are assigned by the methods SetCurrentItemExtraIntProperty 345 and SetCurrentItemExtraStrProperty 346.

Unicode note: names of non-text items are ANSI strings.

See also methods:

- GetCurrentControlInfo 309;
- SetControlInfoEd 341.

See also properties:

- CurItemNo 288;
- CurItemStyle 289.

See also:

- Modifying RichView items 51;
- Item types 29;
- "Tags" 34.

1.4.1.2.2.99 TRichView Edit.SetCurrentHotspotInfo

Modifies the item at the position of caret, if this item is a *hotspot* 97.

```
procedure SetCurrentHotspotInfo(const AName: TRVAnsiString 623;
  AImageIndex, AHotImageIndex: Integer; AImageList: TCustomImageList;
  const ATag: TRVTag 645);
```

SetCurrentHotspotInfo(...) is equivalent to TopLevelEditor 295.**SetHotspotInfoEd** 349(TopLevelEditor 295.CurItemNo 288, ...).

This method can be used if CurItemStyle 289 returns *rvsHotspot* 663.

Parameters:

AName – name of *hotspot* 97, any string without line break (CR, LF) characters. It can also be set using SetCurrentItemText 347 method.

AImageList – not used, reserved, set it to *nil*..

AImageIndex – index of image in the image list. It can also be set using SetCurrentItemExtraIntPropertyEx 345 method.

AHotImageIndex – index of "hot" image in the image list. This image is displayed under the mouse pointer (in TRichView, or in TRichViewEdit in hypertext mode), or when user moves the caret to this item (in TRichViewEdit). It can also be set using SetCurrentItemExtraIntPropertyEx 345 method.

ATag – *tag* of the item. You can use value returned by GetCurrentHotspotInfo 310 or GetCurrentTag 316 for this item. The *tag* can also be set by SetCurrentTag 349 method.

Method type: editing-style.

Additional item properties are assigned by the methods SetCurrentItemExtraIntProperty 345 and SetCurrentItemExtraStrProperty 346.

Unicode note: names of non-text items are ANSI strings.

See also methods:

- GetCurrentHotspotInfo 310;
- SetHotspotInfoEd 349.

See also properties:

- CurItemNo 288;
- CurItemStyle 289.

See also:

- Modifying RichView items 51;
- Item types 29;
- "Tags" 34.

1.4.1.2.2.100 TRichView Edit.SetCurrentItemExtraIntProperty -Ex

The methods change value of an integer property for the item at the position of caret.

```
procedure SetCurrentItemExtraIntProperty(Prop: TRVExtraItemProperty 627;
  Value: Integer; AutoReformat: Boolean);
procedure SetCurrentItemExtraIntPropertyEx(Prop, Value: Integer;
  AutoReformat: Boolean);
```

(introduced in versions 1.7 and 15)

SetCurrentItemExtraIntProperty[Ex](...) is equivalent to TopLevelEditor 295.**SetItemExtraIntProperty[Ex]Ed** 350(TopLevelEditor 295.CurItemNo 288, ...).

These methods set a new **Value** of the item's property identified by **Prop**.

In **SetCurrentItemExtraIntProperty**, **Prop**'s type is TRVExtraItemProperty 627. See information about this type for the list of properties.

In **SetCurrentItemExtraIntPropertyEx**, **Prop**'s type is Integer. If **Prop** can be converted to TRVExtraItemProperty, **SetCurrentItemExtraIntPropertyEx** works like **SetCurrentItemExtraIntProperty**. In addition, it supports properties identified by rveipc*** constants 658

If **AutoReformat**=*True*, these methods automatically reformat the document and call OnChange 357.

If **AutoReformat**=*False* (this can be useful if you set several properties of an item at once), call Change 303 and reformat the document after it yourself. A reformatting depends on the property:

- some properties do not require reformatting (for example, colors), repainting is enough (call Invalidate);
- some properties require a local reformatting (change properties between calls of BeginItemModify 301 and EndItemModify 307);
- some properties require a complete reformatting (for example, "start-from" and "reset" of numbered sequences) (call Reformat 214).

Method type: editing-style.

See also methods:

- GetCurrentItemExtraIntProperty[Ex] 312;
- SetItemExtraIntProperty[Ex]Ed 350;
- SetCurrentItemExtra 346**Str** 346Property 346;

See also methods of TCustomRichView:

- SetItemExtraIntProperty[Ex] 230.

1.4.1.2.2.101 TRichView Edit.SetCurrentItemExtraStrProperty -Ex

The methods change a value of a string property for the item at the position of caret

```
procedure SetCurrentItemExtraStrProperty(Prop: TRVExtraItemStrProperty 630;
  const Value: String; AutoReformat: Boolean);
procedure SetCurrentItemExtraStrPropertyEx(Prop: Integer;
  const Value: String; AutoReformat: Boolean);
```

(introduced in versions 1.9 and 15)

SetCurrentItemExtraStrProperty[Ex](...) is equivalent to TopLevelEditor 295.**SetItemExtraStrProperty[Ex]Ed** 351(TopLevelEditor 295.CurItemNo 288, ...).

This method sets a new **Value** of the item's property identified by **Prop**.

In **SetCurrentItemExtraStrProperty**, **Prop**'s type is TRVExtraItemStrProperty 630. See information about this type for the list of properties.

In **SetCurrentItemExtraStrPropertyEx**, **Prop**'s type is Integer. If **Prop** can be converted to TRVExtraItemStrProperty, **SetCurrentItemExtraStrPropertyEx** works like **SetCurrentItemExtraStrProperty**. In addition, it supports properties identified by rvespc*** constants 661

If **AutoReformat**=*True*, these methods automatically reformat the document and call OnChange 357.

If **AutoReformat**=*False* (this can be useful if you set several properties of an item at once), call Change 303 and reformat the document after it yourself. A reformatting depends on the property:

- some properties do not require reformatting or repainting;
- some properties require a local reformatting (for example, "format string" of numbered sequences) (change properties between calls of BeginItemModify 301 and EndItemModify 307);
- some properties require a complete reformatting (for example, "sequence name" of numbered sequences) (call Reformat 214).

Method type: editing-style.

Unicode note: string properties are Unicode strings for Delphi/C++Builder 2009 or newer, and ANSI strings

for older versions of Delphi/C++Builder.

See also methods:

- GetCurrentItemExtraStrProperty[Ex] 313;
- SetItemExtraStrProperty[Ex]Ed 351;
- SetCurrentItemExtra 345**Int** 345Property[Ex] 345.

See also methods of TCustomRichView:

- SetItemExtraStrProperty[Ex] 231.

1.4.1.2.2.102 TRichView Edit.SetCurrentItemText -A -W

Changes text of text item or a name of non-text item at the position of caret.

```
procedure SetCurrentItemText(const s: String);
procedure SetCurrentItemTextA(const s: TRVAnsiString 623);
procedure SetCurrentItemTextW(const s: TRVUnicodeString 647);
```

(Introduced in version 1.4, 1.7)

- **SetCurrentItemText**(s) is equivalent to TopLevelEditor 295.**SetItemTextEd** 352(TopLevelEditor 295.CurItemNo 288, s).
- **SetCurrentItemTextA**(s) is equivalent to TopLevelEditor 295.**SetItemTextEdA** 352(TopLevelEditor 295.CurItemNo 288, s).
- **SetCurrentItemTextW**(s) is equivalent to TopLevelEditor 295.**SetItemTextEdW** 352(TopLevelEditor 295.CurItemNo 288, s).

Parameters:

s – text assigned to the item. For text items, this is a visible text. For non-text item, this is a string value associated with the item (not displayed). **S** must not contain line break (CR and LF) characters. For text items, it must not contain CR, LF, TAB, FF characters (#13, #10, #9, #12).

Unicode notes:

SetCurrentItemTextA sets text from ANSI string:

- If the item is a non-text item or a text item of ANSI style, the text is assigned as it is.
- If the item is a Unicode text item, the function converts **s** to Unicode (conversion is based on Charset 434 of the item style).

SetCurrentItemTextW sets text from Unicode string:

- If the item is a Unicode text item, the text is assigned as it is.
- If the item is a non-text item or a text item of ANSI style, the function converts **s** to ANSI (conversion is based on Charset 434 of the item style, or on RVStyle.DefCodePage 388 for non-text items).

SetCurrentItemText works like **SetCurrentItemTextW** in Delphi/C++Builder 2009 or newer, and like **SetCurrentItemTextA** in the older versions of Delphi/C++Builder.

Methods type: editing-style.

See also methods:

- GetCurrentItemText -A -W 313;
- SetItemTextEd -A -W 352.

See also methods of TCustomRichView:

- SetItemText -A -W 232.

See also properties:

- CurItemNo 288;
- CurItemStyle 289.

See also:

- Obtaining RichView items 50;
- Item types 29;
- "Tags" 34;
- Unicode in RichView 64.

1.4.1.2.2.103 TRichView Edit.SetCurrentItemVAlign

Changes vertical alignment (position relative to the line) of the item at the position of caret.

```
procedure SetCurrentItemVAlign(VAlign: TRVVAlign 648);
```

(introduced in version 12)

SetCurrentItemVAlign(VAlign) is equivalent to TopLevelEditor 295.**SetItemVAlignEd** 353(TopLevelEditor 295.CurItemNo 288, VAlign).

Method type: editing-style.

This method must not be called if the current item is one of: text items 88, tables 98, *breaks* 93, tabs 89, list markers 99, footnotes 104 and endnotes 103.

See also methods:

- GetCurrentItemVAlign 314;
- SetItemVAlignEd 353.

See also properties:

- CurItemNo 288.

See also:

- Modifying RichView items 51.

1.4.1.2.2.104 TRichView Edit.SetCurrentPictureInfo

Modifies the item at the position of caret, if this item is a picture 90 or *hot-picture* 93.

```
procedure SetCurrentPictureInfo(const AName: TRVAnsiString 623;
  Agr: TGraphic; AVAlign: TRVVAlign 648; const ATag: TRVTag 645);
```

SetCurrentPictureInfo(...) is equivalent to TopLevelEditor 295.**SetPictureInfoEd** 354(TopLevelEditor 295.CurItemNo 288, ...).

This method can be used if CurItemStyle 289 returns *rvsPicture* 663.

Parameters:

AName – name of the item, any string without line break (CR, LF) characters. It can also be set using SetCurrentItemText 347 method.

Agr – graphic object. If this item does not use a shared image (by default), **Agr** must be an unique graphic object (you cannot assign the same graphic object to two or more items) or the value returned by GetCurrentPictureInfo 315 (the image is shared if *rvepShared* extra item property 627 is nonzero). If this item does not use a shared image, and **Agr** is not equal to the value returned by GetCurrentPictureInfo 315, the previously used graphic object is freed.

AVAlign – vertical alignment of the picture.

ATag – *tag* of the item. You can use value returned by GetCurrentPictureInfo 315 or GetCurrentTag 316 for this item. The *tag* can also be set by SetCurrentTag 349 method.

Method type: editing-style.

Additional item properties are assigned by the methods SetCurrentItemExtraIntProperty 345 and

SetCurrentItemExtraStrProperty 346.

Unicode note: names of non-text items are ANSI strings.

See also methods:

- GetCurrentPictureInfo 315;
- SetPictureInfoEd 354;
- SetCurrentItemExtraIntProperty 345.

See also properties:

- CurItemNo 288;
- CurItemStyle 289.

See also:

- Modifying RichView items 51;
- Item types 29;
- "Tags" 34.

1.4.1.2.2.105 TRichView Edit.SetCurrentTag

Changes tag of the item at the position of caret.

```
procedure SetCurrentTag(ATag: TRVTag 645);
```

SetCurrentTag(Tag) is equivalent to TopLevelEditor 295.**SetItemTagEd** 352(TopLevelEditor 295.CurItemNo 288, Tag).

Method type: editing-style.

See also methods:

- GetCurrentTag 316;
- SetItemTagEd 352.

See also properties:

- CurItemNo 288.

See also:

- Modifying RichView items 51;
- "Tags" 34.

1.4.1.2.2.106 TRichView Edit.SetHotspotInfoEd

An editing-style method for changing properties of the **ItemNo**-th item of RichView, if this item is a *hotspot* 97.

```
procedure SetHotspotInfoEd(ItemNo: Integer; const AName: TRVAnsiString 623;
  AImageIndex,  AHotImageIndex: Integer;
  AImageList: TCustomImageList; const ATag: TRVTag 645);
```

Parameters:

ItemNo – index of the item. The item must be of *hotspot* 97 type (*rvsHotspot* 663), otherwise the method raises ERichViewError 608 exception. Items are indexed from 0 to ItemCount 144-1, GetItemStyle 194 returns type of item.

Items of subdocuments (table cells 567) are not included in the items range of the main document. For items in cells, activate inplace-editor, select the proper item in the cell (SelSelectionBounds 235), and call SetHotspotInfoEd of this inplace-editor. Alternatively, you can use SetCurrentHotspotInfo 345.

AName – name of *hotspot* 97, any string without line break (CR, LF) characters. It can also be set using SetItemTextEd 352 method.

AImageList – not used, reserved, set it to *nil*.

AImageIndex – index of image in image list. It can also be set using SetItemExtraIntPropertyExEd 350 method.

AHotImageIndex – index of "hot" image in image list. This image is displayed under the mouse pointer (in TRichView, or in TRichViewEdit in hypertext mode), or when user moves the caret to this item (in TRichViewEdit). It can also be set using SetItemExtraIntPropertyExEd 350 method.

ATag – *tag* of the item. You can use value returned by GetHotspotInfo 189 or GetItemTag 195 for this item. The *tag* can also be set by SetItemTagEd 352 method.

Method type: editing-style (unlike SetHotspotInfo 229).

Additional item properties are assigned by the methods SetItemExtraIntPropertyEd 350 and SetItemExtraStrPropertyEd 351.

Unicode note: names of non-text items are ANSI strings.

When possible, use SetCurrentHotspotInfo 345 instead of this method.

See also methods of TCustomRichView:

- GetHotspotInfo 189;
- GetItemStyle 194;
- SetHotspotInfo 229.

See also methods:

- SetCurrentHotspotInfo 345.

See also properties:

- ItemCount 144.

See also:

- Modifying RichView items 51;
- Item types 29;
- "Tags" 34.

1.4.1.2.2.107 TRichView Edit.SetItemExtraIntProperty-Ed -ExEd

Editing-style methods for changing a value of the specified additional integer property of the **ItemNo**-th item..

```
procedure SetItemExtraIntPropertyEd(ItemNo: Integer;
  Prop: TRVExtraItemProperty 627; Value: Integer;
  AutoReformat: Boolean);
procedure SetItemExtraIntPropertyExEd(ItemNo, Prop,
  Value: Integer; AutoReformat: Boolean);
```

(introduced in versions 1.7 and 15)

These methods set a new **Value** of the item property identified by **Prop**.

ItemNo – index of the item, from 0 to ItemCount 144-1. Items of subdocuments (table cells 567) are not included in the items range of the main document. For items in cells, activate inplace-editor, select the proper item in the cell (SelSelectionBounds 235), and call SetItemExtraIntProperty[Ex]Ed of this inplace-editor. Alternatively, you can use SetCurrentItemExtraIntProperty[Ex] 345.

In **SetItemExtraIntPropertyEd**, **Prop**'s type is TRVExtraItemProperty 627. See information about this type for the list of properties.

In **SetItemExtraIntPropertyExEd**, **Prop**'s type is Integer. If **Prop** can be converted to TRVExtraItemProperty, **SetItemExtraIntPropertyExEd** works like **SetItemExtraIntPropertyEd**. In addition, it supports properties identified by rveipc*** constants 658

If **AutoReformat**=*True*, these methods automatically reformat the document and call OnChange 357.

If **AutoReformat**=*False* (this can be useful if you set several properties of an item at once), call Change[303] and reformat the document after it yourself. A reformatting depends on the property:

- some properties do not require reformatting (for example, colors), repainting is enough (call Invalidate);
- some properties require a local reformatting (change properties between calls of BeginItemModify[301] and EndItemModify[307]);
- some properties require a complete reformatting (for example, "start-from" and "reset" of numbered sequences) (call Reformat[214]).

Methods type: editing-style (unlike SetItemExtraIntProperty[Ex][230]).

When possible, use SetCurrentItemExtraIntProperty[Ex][345] instead of these methods.

See also methods:

- SetItemExtra[351]**Str**[351]PropertyEd[351];
- SetCurrentItemExtraIntProperty[Ex][345].

See also methods of TCustomRichView:

- SetItemExtraIntProperty[Ex][230].

1.4.1.2.2.108 TRichView Edit.SetItemExtraStrProperty-Ed -ExEd

Editing-style methods for changing a value of the specified additional string property of the **ItemNo**-th item..

```
procedure SetItemExtraStrPropertyEd(ItemNo: Integer;
  Prop: TRVExtraItemStrProperty[630]; const Value: String;
  AutoReformat: Boolean);
procedure SetItemExtraStrPropertyExEd(ItemNo, Prop: Integer;
  const Value: String; AutoReformat: Boolean);
```

(introduced in versions 1.9 and 15)

These methods set a new **Value** of the item property identified by **Prop**.

In **SetItemExtraStrPropertyEd**, **Prop**'s type is TRVExtraItemStrProperty[630]. See information about this type for the list of properties.

In **SetItemExtraStrPropertyExEd**, **Prop**'s type is Integer. If **Prop** can be converted to TRVExtraItemStrProperty, **SetItemExtraStrPropertyExEd** works like **SetItemExtraStrPropertyEd**. In addition, it supports properties identified by rvespc*** constants[661]

ItemNo – index of the item, from 0 to ItemCount[144]-1.

Items of subdocuments (table cells[567]) are not included in the items range of the main document. For items in cells, activate inplace-editor, select the proper item in the cell (SelSelectionBounds[235]), and call SetItemExtraStrProperty[Ex]Ed of this inplace-editor. Alternatively, you can use SetCurrentItemExtraStrProperty[346].

If **AutoReformat**=*True*, these methods automatically reformat the document and call OnChange[357].

If **AutoReformat**=*False* (this can be useful if you set several properties of an item at once), call Change[303] and reformat the document after it yourself. A reformatting depends on the property:

- some properties do not require reformatting or repainting;
- some properties require a local reformatting (for example, "format string" of numbered sequences) (change properties between calls of BeginItemModify[301] and EndItemModify[307]);
- some properties require a complete reformatting (for example, "sequence name" of numbered sequences) (call Reformat[214]).

Method type: editing-style (unlike SetItemExtraStrProperty[Ex][231]).

When possible, use SetCurrentItemExtraStrProperty[Ex][346] instead of this method.

Unicode note: string properties are Unicode strings for Delphi/C++Builder 2009 or newer, and ANSI strings for older versions of Delphi/C++Builder.

See also methods:

- SetCurrentItemExtraStrProperty[Ex] 346;
- SetItemExtra 350 Int 350 Property[Ex]Ed 350.

See also methods of TCustomRichView:

- SetItemExtraStrProperty[Ex] 231.

1.4.1.2.2.109 TRichView Edit.SetItemTagEd

Changes tag of the **ItemNo**-th item.

```
procedure SetItemTagEd(ItemNo: Integer; const ATag: TRVTag 645);
```

Items are indexed from 0 to ItemCount 144-1.

Items of subdocuments (table cells 567) are not included in the items range of the main document. For items in cells, activate inplace-editor, select the proper item in the cell (SelSelectionBounds 235), and call SetItemTagEd of this inplace-editor. Alternatively, you can use SetCurrentTag 349.

Tag can be an integer value or a pointer to dynamically allocated string, see *Tags* 34.

Method type: editing-style, unlike SetItemTag 231.

When possible, use SetCurrentTag 349 instead of this method.

See also methods of TCustomRichView:

- GetItemTag 195;
- SetItemTag 231.

See also methods:

- SetCurrentTag 349.

See also properties:

- ItemCount 144.

See also:

- Modifying RichView items 51;
- "Tags" 34.

1.4.1.2.2.110 TRichView Edit.SetItemTextEd -A -W

Editing-methods for changing text of text item or name of non-text item.

```
procedure SetItemTextEd(ItemNo: Integer; const s: String);
procedure SetItemTextEdA(ItemNo: Integer; const s: TRVAnsiString 623);
procedure SetItemTextEdW(ItemNo: Integer; const s: TRVUnicodeString 647);
```

(Introduced in version 1.4, 1.7)

Parameters:

ItemNo – index of the item. Items are indexed from 0 to ItemCount 144-1, GetItemStyle 194 returns type of item.

Items of subdocuments (table cells 567) are not included in the items range of the main document. For items in cells, activate inplace-editor, select the proper item in the cell (SelSelectionBounds 235), and call SetItemTextEd* of this inplace-editor. Alternatively, you can use SetCurrentItemText* 347.

s – text assigned to the item. For text items, this is a visible text. For non-text item, this is a string value associated with the item (not displayed). **S** must not contain line break (CR and LF) characters. For text items, it must not contain CR, LF, TAB, FF characters (#13, #10, #9, #12).

Unicode notes:

SetItemTextEdA sets text from ANSI string:

- If the item is a non-text item or a text item of ANSI style, the text is assigned as it is.
- If the item is a Unicode text item, the function converts **s** to Unicode (conversion is based on Charset 434 of the item style).

SetItemTextEdW sets text from Unicode string:

- If the item is a Unicode text item, the text is assigned as it is.
- If the item is a non-text item or a text item of ANSI style, the function converts **s** to ANSI (conversion is based on Charset 434 of the item style, or on RVStyle.DefCodePage 388 for non-text items).

SetItemTextEd works like **SetItemTextEdW** in Delphi/C++Builder 2009 or newer, and like **SetItemTextEdA** in the older versions of Delphi/C++Builder.

Methods type: editing-style.

See also methods:

- SetCurrentItemText -A -W 347.

See also methods of TCustomRichView:

- GetItemText -A -W 195;
- SetItemText -A -W 232.

See also properties:

- ItemCount 144.

See also:

- Modifying RichView items 51;
- "Tags" 34;
- Unicode in RichView 64.

1.4.1.2.2.111 TRichView Edit.SetItemVAlignEd

Changes vertical alignment (position relative to the line) of the **ItemNo**-th item.

```
procedure SetItemVAlignEd(ItemNo: Integer; VAlign: TRVVAlign 648);
```

(introduced in version 12)

Items are indexed from 0 to ItemCount 144-1.

Items of subdocuments (table cells 567) are not included in the items range of the main document. For items in cells, activate inplace-editor, select the proper item in the cell (SelSelectionBounds 235), and call SetItemVAlignEd of this inplace-editor. Alternatively, you can use SetCurrentItemVAlign 348.

Method type: editing-style, unlike SetItemVAlign 233.

This method must not be called for text items 88, tables 98, *breaks* 93, tabs 89, list markers 99, footnotes 104 and endnotes 103.

When possible, use SetCurrentItemVAlign 348 instead of this method.

See also methods of TCustomRichView:

- GetItemVAlign 196;
- SetItemVAlign 233.

See also methods:

- SetCurrentItemVAlign 348.

See also properties:

- ItemCount 144.

See also:

- Modifying RichView items [51].

1.4.1.2.2.112 TRichView Edit.SetPictureInfoEd

An editing-style method for changing properties of the **ItemNo**-th item of RichView, if this item is a picture [90] or *hot-picture* [93].

```
procedure SetPictureInfoEd(ItemNo: Integer; const AName: TRVAnsiString [623];
  Agr: TGraphic; AVAlign: TRVVAlign [648]; const ATag: TRVTag [645]);
```

Parameters:

ItemNo – index of the item. The item must be of picture [90] or *hot-picture* [93] type (*rvsPicture or rvsHotPicture* [663]), otherwise the method raises ERichViewError [608] exception. Items are indexed from 0 to ItemCount [144]-1, GetItemStyle [194] returns type of item.

Items of subdocuments (table cells [567]) are not included in the items range of the main document. For items in cells, activate inplace-editor, select the proper item in the cell (SelSelectionBounds [235]), and call SetPictureInfoEd of this inplace-editor. Alternatively, you can use SetCurrentPictureInfo [348].

AName – name of the item, any string without line break (CR, LF) characters. It can also be set using SetItemTextEd [352] method.

Agr – graphic object. If this item does not use a shared image (by default), **Agr** must be an unique graphic object (you cannot assign the same graphic object to two or more items) or the value returned by GetPictureInfo [201] for this item (the image is shared if *rvepShared* extra item property [627] is nonzero). If this item does not use a shared image, and **Agr** is not equal to the value returned by GetPictureInfo [201] for this item, the previously used graphic object is freed.

AVAlign – vertical alignment of the picture.

ATag – *tag* of the item. You can use value returned by GetPictureInfo [201] or GetItemTag [195] for this item. The *tag* can also be set by SetItemTagEd [352] method.

Method type: editing-style (unlike SetPictureInfo [234]).

Additional item properties are assigned by the methods SetItemExtraIntPropertyEd [350] and SetItemExtraStrPropertyEd [351].

Unicode note: names of non-text items are ANSI strings.

When possible, use SetCurrentPictureInfo [348] instead of this method.

See also methods of TCustomRichView:

- GetPictureInfo [201];
- GetItemStyle [194];
- SetPictureInfo [234].

See also methods:

- SetCurrentPictureInfo [348].

See also properties:

- ItemCount [144].

See also:

- Modifying RichView items [51];
- Item types [29];
- "Tags" [34].

1.4.1.2.2.113 TRichView Edit.SetUndoGroupMode

Begins or ends group of editing operations which will be undone/redone as a single operation.

```
procedure SetUndoGroupMode(GroupUndo: Boolean);
```

(Introduced in version 1.3)

See Undo in RichViewEdit 54.

See also methods:

- BeginUndoGroup 302;
- BeginUndoCustomGroup 302.

1.4.1.2.2.114 TRichView Edit.Undo

Undoes the last editing operation.

```
procedure Undo;
```

(Introduced in version 1.3)

This method does nothing if there is no action to undo.

Information about the operation to redo is returned by the methods UndoAction 355 and UndoName 355.

This method must be called only when the document is formatted.

This method is called when the user presses Ctrl + Z or Alt + Backspace.

Method type: editing-style.

See also:

- Redo 332;
- Undo in RichViewEdit 54.

1.4.1.2.2.115 TRichView Edit.UndoAction

Returns type of operation to be undone 355 next.

```
function UndoAction: TRVUndoType 647;
```

(Introduced in version 1.3)

See TRVUndoType 647 for the list of operations. If the returned value is *rvutCustom*, you can use UndoName 355.

See also:

- RedoAction 333;
- Undo in RichViewEdit 54.

1.4.1.2.2.116 TRichView Edit.UndoName

Returns name of operation to be undone 355 next.

```
function UndoName: String;
```

(Introduced in version 1.3)

Returns empty string for the standard undo operations, i.e. all operations but *rvutCustom* (see UndoAction 355).

Returns non-empty string only for custom undo operation initiated by BeginUndoCustomGroup 302.

See also:

- RedoName 333;
- Undo in RichViewEdit 54.

1.4.1.2.3 Events

1.4.1.2.3.1 TRichView Edit.OnCaretGetOut

Occurs when user presses arrow keys at the beginning/end of document in "outside" direction.

```
type
  TRVGetOutDirection = (rvdLeft, rvdUp, rvdRight, rvdDown,
    rvdTop, rvdBottom);
  TRVOnCaretGetOutEvent =
    procedure (Sender: TCustomRichViewEdit;
     Direction: TRVGetOutDirection) of object;

property OnCaretGetOut:TRVOnCaretGetOutEvent;
```

(introduced in version 1.4, 1.5)

Direction	The event happens if...
rvdLeft	User pressed **Left** (or **Right**, if BiDiMode[137] =*rvbdRightToLeft*) while the caret is at the beginning of the document
rvdUp	User pressed **Up** while the caret is in the first line of the document
rvdRight	User pressed **Right** (or **Left**, if BiDiMode[137] =*rvbdRightToLeft*) while the caret is at the end of the document
rvdDown	User pressed **Down** while the caret is in the last line of the document.
rvdTop	User pressed **Ctrl**+**Home**
rvdBottom	User pressed **Ctrl**+**End**

This event can be used to move focus to another control.

1.4.1.2.3.2 TRichView Edit.OnCaretMove

Occurs when the caret is moved to the another position within the document

```
property OnCaretMove: TNotifyEvent;
```

(introduced in version 1.5)

Warning: This event may occur when document is not completely formatted! Do not use any methods relying on formatting inside this event, including methods working with selection (for example, do not use SelectionExists[224] here). If you need to update user interface reflecting selection, use OnSelect[265].

In this event, you can use the following:

- CurItemNo[288] property;
- OffsetInCurItem[294] property;
- GetCurrentLineCol[314] method.

1.4.1.2.3.3 TRichView Edit.OnChange

Occurs when document is changed.

```
property OnChange: TNotifyEvent;
```

This event occurs when:

- user edits document, or
- calling any editing-style method, or
- calling Change 303.

Viewer-style methods do not generate this event.

See also property:

- Modified 293.

1.4.1.2.3.4 TRichView Edit.OnChanging

Occurs before editing operations

```
type
  TRVChangingEvent = procedure (Sender: TCustomRichViewEdit;
    var CanEdit: Boolean) of object;

property OnChanging: TRVChangingEvent;
```

Set **CanEdit** to *False* to prevent editing.

This event occurs when:

- user tries to edit text, or
- calling any editing-style method (before changes are actually done).

Viewer-style methods do not generate this event.

See also:

- OnChange 357 event;
- ReadOnly 294 property.

1.4.1.2.3.5 TRichView Edit.OnCheckStickingItems

Allows to "stuck" items together.

```
type
  TRVStickingItemsEvent = procedure (Sender: TCustomRichViewEdit;
    RVData: TCustomRVData 608; FirstItemNo: Integer; var Sticking: Boolean) of object;

property OnCheckStickingItems: TRVStickingItemsEvent;
```

(introduced in version 14)

This event occurs before inserting something after the **FirstItemNo**-th item in **RVData** (if **FirstItemNo**=-1, this insertion is at the beginning of **RVData**).

Assign **Sticking**=*True* to prevent this insertion.

This event occurs after all default checking for sticking is done (see the Protection 439 property of a text style).

1.4.1.2.3.6 TRichView Edit.OnCurParaStyleChanged

Occurs when the index of current paragraph style (value of CurParaStyleNo 289 property) is changed.

```
property OnCurParaStyleChange: TNotifyEvent;
```

This event can occur when user moves caret with mouse or keyboard, or when he/she edits document. Some methods of RichViewEdit can also invoke this event.

Use this event to update user interface in your application.

See also properties:

- CurParaStyleNo 289.

See also events:

- OnCurTextStyleChanged 358.

1.4.1.2.3.7 TRichView Edit.OnCurTextStyleChanged

Occurs when the index of current text style (value of CurTextStyleNo 290 property) is changed.

```
property OnCurTextStyleChange: TNotifyEvent;
```

This event can occur when user moves caret with mouse or keyboard, or when he/she edits document. Some methods of RichViewEdit can also invoke this event.

Use this event to update user interface in your application.

See also properties:

- CurTextStyleNo 290.

See also events:

- OnCurParaStyleChanged 358.

1.4.1.2.3.8 TRichView Edit.OnDraw CustomCaret

Allows to draw a custom caret.

```
type
  TRVDrawCustomCaretEvent = procedure (Sender: TCustomRichViewEdit; Canvas: TCanvas;
    const Rect: TRect) of object;
property OnDrawCustomCaret: TRVDrawCustomCaretEvent;
```

(introduced in version 10)

Custom caret is drawn if CustomCaretInterval 290>0.

OnMeasureCustomCaret 360 event occurs before each call of this event.

Example

This example draws a rectangle/ellipse caret. In combination with the example in OnMeasureCustomCaret 360 topic, it draws rectangle/ellipse around the character to the right of the caret.

```
procedure TMyForm.MyRichViewEditDrawCustomCaret(Sender: TCustomRichViewEdit;
  Canvas: TCanvas; const Rect: TRect);
const f: Boolean=True;
var r: TRect;
begin
  if f then
    Canvas.Pen.Color := clRed
  else
    Canvas.Pen.Color := clBlue;
  f := not f;
```

```
  r := Rect;
  Canvas.Brush.Style := bsClear;
  if f then
    Canvas.Rectangle(r)
  else
    Canvas.Ellipse(r);
end;
```

1.4.1.2.3.9 TRichView Edit.OnDropFiles

Occurs when dropping files in the editor (as a result of drag&drop operation, for example from Windows Explorer)

```
type
  TRVDropFilesEvent =
    procedure (Sender: TCustomRichViewEdit;
      Files: TStrings; var FileAction: TRVDropFileAction;
      var DoDefault: Boolean) of object;
  TRVDropFileAction = (rvdfNone, rvdfInsert, rvdfLink);

property OnDropFiles: TRVDropFilesEvent;
```

(introduced in v1.8)

rvddFiles must be included in AcceptDragDropFormats [287] property, otherwise the editor will not accept files.

Input parameters:

Files – list of names of dropped files.

FileAction = *rvdfNone*.

DoDefault = *True*.

Output parameters:

FileAction – operation that you have performed with these files:

- *rvdfNone* – nothing,
- *rvdfInsert* – the files were inserted,
- rvdfLink – hyperlinks to the files were inserted.

FileAction is ignored if **DoDefault** is set to *True*.

DoDefault – set it to *False* if you have processed the files yourself in this event. By default, RichViewEdit inserts graphic files, *.RTF, *.RVF, *.TXT.

See **the example** [372] (how to insert files of all formats supported by Microsoft Office text import converters) [372].

See also:

- Drag&Drop [55].

1.4.1.2.3.10 TRichView Edit.OnItemResize

Occurs after resizing image [90], control [94] or table [98] with the mouse.

```
type
  TRVItemResizeEvent = procedure(Sender: TCustomRichViewEdit;
    RVData: TCustomRVFormattedData[608]; ItemNo, Val1, Val2: Integer) of object;
property OnItemResize: TRVItemResizeEvent;
```

(introduced in version 10)

Parameters:

RVData and **ItemNo** identify the resized item. **RVData** – document containing this item; it can be **Sender**.RVData[151], table cell[567], or RVData of cell inplace-editor. **ItemNo** – index of this item inside **RVData**.

(**Val1**, **Val2**) are:

- for images and controls: (-1, -1).
- for tables: (0, row index) after resizing row, or (1, column index) after resizing columns.

1.4.1.2.3.11 TRichView Edit.OnItemTextEdit

Occurs when text item[88]'s text is changed as a result of editing operation.

```
type
  TRVItemTextEditEvent = procedure (Sender: TCustomRichViewEdit;
    const OldText: TRVRawByteString[638]; RVData: TCustomRVData[608]; ItemNo: Integer;
    var NewTag: TRVTag[645]; var NewStyleNo: Integer) of object;
property OnItemTextEdit: TRVItemTextEditEvent;
```

(introduced in version 10)

This event is designed to update tags of hyperlinks.

This event is called when document is not completely formatted, so do not use any methods for changing document or methods requiring formatted document in this event.

Input Parameters:

OldText – the item text before the editing operation. For Unicode items, this is a "raw" Unicode text.

RVData, **ItemNo** – identify the item. **RVData** – document containing this item; it can be **Sender**.RVData[151], table cell[567], or RVData of cell inplace-editor. **ItemNo** – index of this item inside **RVData**. You can get text after the editing operation using **RVData**.GetItemTextR[195](**ItemNo**).

NewTag – the item *tag*[34], the same as **RVData**.GetItemTag[195](**ItemNo**)

NewStyleNo – the item style, index in the collection Style[156].TextStyles[404], the same as **RVData**.GetItemStyle[194](**ItemNo**)

Output Parameters:

NewTag – value of *tag*[34] to assign to this item.

NewStyleNo – style index to assign to this item.

1.4.1.2.3.12 TRichView Edit.OnMeasureCustomCaret

Allows to define size and position for custom caret.

```
type
  TRVMeasureCustomCaretEvent = procedure (Sender: TCustomRichViewEdit;
    var Rect: TRect) of object;
property OnMeasureCustomCaret: TRVMeasureCustomCaretEvent;
```

(introduced in version 10)

Custom caret is drawn if CustomCaretInterval[290]>0.

This event is called before each call to OnDrawCustomCaret[358]. It allows to set your own size and position for the caret. The default size and position are the same as of the default system caret. The editor stores background under the **Rect**, it will be restored after the call to OnDrawCustomCaret[358].

Example

This example defines the custom caret width. For non-text items, or at the end of item, it sets the caret width =

default width + 10.For text items [88], it sets the caret width equal to width of character to the right of the caret.

This example uses TMyForm.Canvas to calculate character width.

```
procedure TMyForm.MyRichViewEditMeasureCustomCaret(Sender: TCustomRichViewEdit[282];
  var Rect: TRect);
var s: String;
begin
  Sender := Sender.TopLevelEditor[295];
  if Sender.CurItemStyle[289]<0 then begin
    inc(Rect.Right, 10);
    exit;
  end;
  if Sender.OffsetInCurItem[294]>=Sender.GetOffsAfterItem[200](Sender.CurItemNo[288]) then
  begin
    inc(Rect.Right, 10);
    exit;
  end;
  s := Sender.GetItemText[195](Sender.CurItemNo[288]);
  Canvas.Font.Assign(Sender.Style[156].TextStyles[404][Sender.CurItemStyle[289]]);
  Rect.Right := Rect.Right+Canvas.TextWidth(s[Sender.OffsetInCurItem[294]]);
end;
```

1.4.1.2.3.13 TRichView Edit.OnOleDragEnter

OLE drag&drop event. It occurs when the user drags the mouse pointer into the editor. Indicates whether a drop can be accepted, and, if so, the effect of the drop.

```
type
  TRVOleDragEnterEvent = procedure (Sender: TCustomRichView[128];
    const DataObject: IDataObject; Shift: TShiftState;
    X, Y: Integer; PossibleDropEffects: TRVOleDropEffects[636];
    var DropEffect: TRVOleDropEffect[636]) of object;
property OnOleDragEnter: TRVOleDragEnterEvent;
```

(introduced in version 10)

This is a low level event. Do not use it if you are not familiar with OLE drag&drop.

After calling this event, OnOleDragOver [362] will be called while the user moves the mouse pointer inside the editor. Dragging will be finished either by calling OnOleDrop [363] (successfully) or by calling OnOleDragLeave [362] (canceled, or mouse left the editor).

Input Parameters:

DataObject – dragged data (IDataObject is declared in ActiveX unit).

Shift – keyboard state.

PossibleDropEffects – a list of allowed operations, set of *rvdeCopy*, *rvdeMove*, *rvdeLink*.

DropEffect – operation that the editor would choose by default.

Output Parameters:

DropEffect – operation that you want to perform with the dragged data (one of **PossibleDropEffects**, or *rvdeNone* if you do not want to accept it).

See also events:

- OnOleDragOver [362];
- OnOleDragLeave [362];
- OnOleDrop [363].

See also:

- Drag&drop in TRichView 55.

1.4.1.2.3.14 TRichView Edit.OnOleDragLeave

OLE drag&drop event. It occurs when the user drags the mouse cursor out of the editor or cancels the current drag&drop operation.

```
property OnOleDragLeave: TNotifyEvent;
```

(introduced in version 10)

This is a low level event. Do not use it if you are not familiar with OLE drag&drop.

The first event in sequence is OnOleDragEnter 361, it occurs when the user moves the mouse pointer into the editor while dragging. After that, OnOleDragOver 362 is called while the user moves the mouse inside the editor. Dragging is finished either by calling OnOleDrop 363 (successfully) or by calling this event.

Input Parameters:

DataObject – dragged data (IDataObject is declared in ActiveX unit).

Shift – keyboard state.

PossibleDropEffects – a list of allowed operations, set of *rvdeCopy*, *rvdeMove*, *rvdeLink*.

DropEffect – operation that the editor would choose by default.

Output Parameters:

DropEffect – operation that you want to perform with the dragged data (one of **PossibleDropEffects**, or *rvdeNone* if you do not want to accept it).

See also events:

- OnOleDragEnter 361;
- OnOleDragOver 362;
- OnOleDrop 363.

See also:

- Drag&drop in TRichView 55.

1.4.1.2.3.15 TRichView Edit.OnOleDragOver

OLE drag&drop event. It occurs when the user moves the mouse pointer over the editor. Called only if the dragging was accepted in OnOleDragEnter 361 (or automatically by the editor).

```
type
  TRVOleDragOverEvent = procedure (Sender: TCustomRichView 128;
    Shift: TShiftState; X, Y: Integer;
    PossibleDropEffects: TRVOleDropEffects 636;
    var DropEffect: TRVOleDropEffect 636) of object;
property OnOleDragOver: TRVOleDragOverEvent;
```

(introduced in version 10)

This is a low level event. Do not use it if you are not familiar with OLE drag&drop.

OnOleDragEnter 361 occurs before this event, with DataObject in parameters. Dragging will be finished either by calling OnOleDrop 363 (successfully) or by calling OnOleDragLeave 362 (canceled, or mouse left the editor).

Input Parameters:

DataObject – dragged data (IDataObject is declared in ActiveX unit).

Shift – keyboard state.

X,**Y** – mouse cursor coordinates, relative to the top left corner of **Sender**.

PossibleDropEffects – a list of allowed operations, set of *rvdeCopy*, *rvdeMove*, *rvdeLink*.

DropEffect – operation that the editor would choose by default.

Output Parameters:

DropEffect – operation that you want to perform with the dragged data (one of **PossibleDropEffects**, or *rvdeNone* if you do not want to accept it).

See also events:

- OnOleDragEnter 361;
- OnOleDragLeave 362;
- OnOleDrop 363.

See also:

- Drag&drop in TRichView 55.

1.4.1.2.3.16 TRichView Edit.OnOleDrop

OLE drag&drop event. It occurs when the user completes drag&drop operation into the editor.This event allows you to insert data in your own format.

```
type
  TRVOleDropEvent = procedure (Sender: TCustomRichView;
    const DataObject: IDataObject; Shift: TShiftState; X, Y: Integer;
    PossibleDropEffects: TRVOleDropEffects;
    var DropEffect: TRVOleDropEffect; var DoDefault: Boolean) of object;
property OnOleDrop: TRVOleDropEvent;
```

(introduced in version 10)

This is a low level event. Do not use it if you are not familiar with OLE drag&drop

The first event in sequence is OnOleDragEnter 361, it occurs when the user moves the mouse pointer into the editor while dragging. After that, OnOleDragOver 362 is called while the user moves the mouse inside the editor. Dragging is finished either by calling this event (successfully) or by calling OnOleDragLeave 362 (canceled, or mouse left the editor).

Input Parameters:

DataObject – dragged data (IDataObject is declared in ActiveX unit).

Shift – keyboard state.

X,**Y** – mouse cursor coordinates, relative to the top left corner of **Sender**.

PossibleDropEffects – a list of allowed operations, set of *rvdeCopy*, *rvdeMove*, *rvdeLink*.

DropEffect – operation that the editor would choose by default.

Output Parameters:

DropEffect – the operation that you have performed (one of **PossibleDropEffects**, or *rvdeNone* if you canceled the insertion). If it is *rvdeMove*, it instructs to delete the dragged object from its source location. The value of this parameter is used only if you set **DoDefault**=*False*, otherwise the editor does the default processing. If you inserted some data in editor in this event, they are automatically selected afterwards.

Limitation: when dragging from the same editor, it decides by itself whether it should move or copy data, and it

ignores **DropEffect**.

See also events:

- OnOleDragEnter[361];
- OnOleDragOver[362];
- OnOleDragLeave[362];
- OnDropFiles[359].

See also:

- Drag&drop in TRichView'[55];
- How to make one line, plain text editor[669].

1.4.1.2.3.17 TRichView Edit.OnParaStyleConversion

Occurs while executing ApplyParaStyleConversion[298] method.

```
type
  TRVStyleConversionEvent =
    procedure (Sender: TCustomRichViewEdit;
      StyleNo, UserData: Integer; AppliedToText: Boolean;
      var NewStyleNo: Integer) of object;

property OnParaStyleConversion:TRVStyleConversionEvent;
```

(introduced in version 1.7)

This event allows you to create custom conversion procedure for styles of the selected paragraphs. This event is called for all selected items allowing to change their paragraph style.

It can be useful for implementing commands like "change paragraph alignment", "change paragraph indents", etc.

Parameters:

StyleNo – current paragraph style (index in collection of styles, Style[156].ParaStyles[398]);

UserData – value passed as a parameter in ApplyParaStyleConversion[298];

AppliedToText – not used

NewStyleNo – initially equals to StyleNo; assign a new value to this parameter to change paragraph style.

Example:

```
procedure TMyForm.MyRichViewEditParaStyleConversion(Sender: TCustomRichViewEdit;
  StyleNo, UserData: Integer; AppliedToText: Boolean;
  var NewStyleNo: Integer);
var ParaInfo: TParaInfo[448];
begin
  // creating paragraph style with the desired attributes
  ParaInfo := TParaInfo.Create(nil);
  try
    ParaInfo.Assign(RVStyle1.ParaStyles[398][StyleNo]);
    case UserData of
      PARA_ALIGNMENT:
        ParaInfo.Alignment := GetAlignment;
      PARA_INDENTINC:
        begin
          ParaInfo.LeftIndent := ParaInfo.LeftIndent+20;
```

```
          if ParaInfo.LeftIndent>1000 then
            ParaInfo.LeftIndent := 1000;
        end;
      PARA_INDENTDEC:
        begin
          ParaInfo.LeftIndent := ParaInfo.LeftIndent-20;
          if ParaInfo.LeftIndent<0 then
            ParaInfo.LeftIndent := 0;
        end;
      PARA_COLOR:
        ParaInfo.Background.Color := ColorDialog1.Color;
      // add your code here....
    end;
    // searching for the style (or adding it if necessary)
    NewStyleNo := RVStyle1.FindParaStyle(ParaInfo);
  finally
    ParaInfo.Free;
  end;
end;
```

This example shows how to implement the following commands:

- change paragraph alignment,
- increase left indent,
- decrease left indent,
- change paragraph background color.

In this code:

- PARA_*** – user-defined integer constants with unique values; these constants identify commands; they are passed to ApplyParaStyleConversion[298] as UserData;
- RVStyle1 – TRVStyle[385] component linked with the MyRichViewEdit;
- ColorDialog1: TColorDialog;
- GetAlignment – function returning paragraph alignment chosen by the user.

You can see that for new paragraph styles added in this event, Standard[431] property is set to *False*. It is important, otherwise DeleteUnusedStyles[181] will not be able to delete this style, even if all paragraphs of this style will be removed.

See demos:

- Demos\Delphi\Editors\Editor 2\
- Demos\CBuilder\Editors\Editor 2\

See also:

- OnStyleConversion[366].

See also methods:

- ApplyParaStyleConversion[298].

1.4.1.2.3.18 TRichView Edit.OnPaste

Allows to paste data from the Clipboard in custom formats.

```
type
  TRVPasteEvent =
    procedure (Sender: TCustomRichViewEdit;
               var DoDefault: Boolean) of object;

property OnPaste: TRVPasteEvent;
```

(introduced in version 1.3)

Occurs before any data are pasted in editor (Paste[328] method, or Ctrl + V / Shift + Insert).

You can insert data in your own format here and/or forbid default pasting procedure (set **DoDefault** to *False*).

Note: see standard priorities of formats for pasting in Paste[328] method.

Example (allowing to paste only one line of plain text):

```
procedure TMyForm.MyRichViewEditPaste(Sender: TCustomRichViewEdit;
  var DoDefault: Boolean);
var s: String;
begin
  s := Clipboard.AsText;
  if (Pos(#10,s)=0) and (Pos(#13,s)=0) then
    MyRichViewEdit.InsertText[327](s,False);
  DoDefault := False;
end;
```

See also:

- RichView and Clipboard[48];
- How to make a plain text editor[669].

1.4.1.2.3.19 TRichView Edit.OnSmartPopupClick

Occurs when user clicks "smart popup"[368] button.

```
type
  TRVSmartPopupClickEvent = procedure (Sender: TCustomRichView;
    Button: TCustomControl) of object;
property OnSmartPopupClick: TRVSmartPopupClickEvent;
```

(introduced in version 10)

This event allows implementing your own effect when clicking "smart popup" button (or on pressing ShortCut[371]). If you use menu, it's not necessary to process this event.

1.4.1.2.3.20 TRichView Edit.OnStyleConversion

Occurs while executing ApplyStyleConversion[299] method.

```
type
  TRVStyleConversionEvent =
    procedure (Sender: TCustomRichViewEdit;
     StyleNo, UserData: Integer; AppliedToText: Boolean;
     var NewStyleNo: Integer) of object;

property OnStyleConversion:TRVStyleConversionEvent;
```

This event allows you to create custom conversion procedure of text styles for the selected text. This event is called for all selected text items allowing to change their text style.

It can be useful to implement commands like "make bold", "apply font", "change text color", etc.

Parameters:

StyleNo – current text style (index in the collection of styles, Style[156].TextStyles[404]);

UserData – value passed as a parameter in ApplyStyleConversion[299];

AppliedToText – *True*, if this event was called for conversion of style of text item; *False*, if it was called for conversion of the current text style (CurTextStyleNo[290]);

NewStyleNo – initially equals to **StyleNo**; assign a new value to this parameter to change text style.

Example 1

```
procedure TMyForm.MyRichViewEditStyleConversion(Sender: TCustomRichViewEdit;
  StyleNo, UserData: Integer; AppliedToText: Boolean;
  var NewStyleNo: Integer);
begin
   if not (AppliedToText and
      (rvprStyleProtect in Style.TextStyles[StyleNo].Protection[439])) then
    NewStyleNo := UserData;
end;
```

This example does exactly the same work as ApplyTextStyle[300], i.e. MyRichViewEdit.ApplyStyleConversion[299] (StyleNo) will be equivalent to MyRichViewEdit.ApplyTextStyle[300](StyleNo).

You can see that ApplyStyleConversion ignores protection of styles. If you wish to protect them, you should check protection yourself and do not change this style (like in the example above).

Example 2

```
procedure TMyForm.MyRichViewEditStyleConversion(Sender: TCustomRichViewEdit;
                  StyleNo, UserData: Integer;
                  AppliedToText: Boolean;
                  var NewStyleNo: Integer);
var FontInfo: TFontInfo[444];
begin
  // creating text style with the desired attributes
  FontInfo := TFontInfo.Create(nil);
  try
    FontInfo.Assign(RVStyle1.TextStyles[404][StyleNo]);
    case UserData of
      TEXT_BOLD:
        if btnBold.Down then
          FontInfo.Style := FontInfo.Style[441]+[fsBold]
        else
          FontInfo.Style := FontInfo.Style-[fsBold];
      TEXT_ITALIC:
        if btnItalic.Down then
          FontInfo.Style := FontInfo.Style+[fsItalic]
        else
          FontInfo.Style := FontInfo.Style-[fsItalic];
      TEXT_UNDERLINE:
        if btnUnderline.Down then
          FontInfo.Style := FontInfo.Style+[fsUnderline]
        else
          FontInfo.Style := FontInfo.Style-[fsUnderline];
      TEXT_APPLYFONTNAME:
        FontInfo.FontName := FontName;
      TEXT_APPLYFONTSIZE:
        FontInfo.Size     := FontSize;
      TEXT_APPLYFONT:
        FontInfo.Assign(FontDialog1.Font);
      TEXT_COLOR:
        FontInfo.Color := ColorDialog1.Color;
      TEXT_BACKCOLOR:
        FontInfo.BackColor := ColorDialog1.Color;
      // add your code here....
    end;
    // searching for the style (or adding it if necessary)
```

```
    NewStyleNo := RVStyle1.FindTextStyle(FontInfo);
  finally
    FontInfo.Free;
  end;
end;
```

This example shows how to implement the following commands:

- make bold/not bold,
- make italic/not italic,
- make underlined/not underlined,
- change font name,
- change font size,
- apply font,
- change text color,
- change text background color.

In this code:

- TEXT_*** – user defined integer constants with unique values; these constants identify commands; they are passed to ApplyStyleConversion as UserData;
- RVStyle1 – TRVStyle component linked with the richview editor;
- ColorDialog1: TColorDialog;
- FontDialog1: TFontDialog;
- btnBold, btnItalic, btnUnderline: TSpeedButton;
- FontName: String;
- FontSize: Integer.

See demos:

- Demos\Delphi\Editors\Editor 2\
- Demos\CBuilder\Editors\Editor 2\

See also:

- OnParaStyleConversion.

See also methods:

- ApplyStyleConversion.

1.4.1.2.4 Classes of Properties

1.4.1.2.4.1 TRVSmartPopupProperties

Properties of "smart popup" buttons (analogs of buttons used for Microsoft Office "smart tags"). Class of TCustomRichViewEdit.SmartPopupProperties.

Unit RVPopup.

Syntax

```
  TRVSmartPopupProperties = class (TPersistent)
```

(introduced in version 10)

Hierarchy

TObject
TPersistent

Using

"Smart popups" allow to edit properties of the current (at the position of caret) item. Of course, you can use popup menu instead, but in some cases "smart popups" may be useful (for example, to show to user that some special commands available for the current item). Only one "smart popup" button can be displayed at the same time.

Use TCustomRichViewEdit.OnCaretMove[356] event to set properties of "smart popup" and display the button. The button is displayed when you assign True to TCustomRichViewEdit.SmartPopupVisible[295]. The editor hides this button automatically on document change or clearing. Since "smart popup" works only for the current item, you must ensure that it is displayed for the current item (update or hide it when the caret moves to another item).

The following properties define visual appearance of "smart popup" button:

- ButtonType[369];
- Color[369], LineColor[370];
- HoverColor[370], HoverLineColor[370];
- ImageList[370], ImageIndex[370].

Hint[370] is displayed when the user moves the mouse pointer over the button.

Position[371] defines the button position.

When the user clicks the button or presses ShortCut[371], Menu[371] is displayed (if assigned). Instead of a menu, you can process TCustomRichViewEdit.OnSmartPopupClick[366] event to execute a command or display a dialog.

Defines how popup buttons look like in hot (under the mouse pointer) state.

```
type
  TRVSmartPopupType = (rvsptDropDown, rvsptShowDialog, rvsptSimple);
property ButtonType: TRVSmartPopupType;
```

Note: TRVSmartPopupType is defined in RVScroll unit.

Type	Meaning	Example
rvsptDropDown	Button width is increased, a triangle is shown. Recommended for buttons displaying menu[371] or popup windows.	
rvsptShowDialog	Button width is increased, an ellipsis is shown. Recommended for buttons displaying dialogs.	
rvsptSimple	Button width is not changed. Recommended for buttons executing an immediate command.	

You can change this property in TCustomRichViewEdit.OnCaretMove[356] (to create the proper button for the item at the position of caret)

Default value:

rvsptDropDown

Background color of a button.

```
property Color: TColor;
```

Default value:

clWindow

See also:

- LineColor[370];
- HoverColor[370];
- HoverLineColor[370].

Contains the text string that can appear when the user moves the mouse over a button.

```
property Hint: String;
```

You can change this property in TCustomRichViewEdit.OnCaretMove[356] (to create the proper button for the item at the position of caret)

Background color of a button when it is below the mouse pointer.

```
property HoverColor: TColor;
```

Default value:

clInfoBk

See also:

- Color[369];
- LineColor[370];
- HoverLineColor[370].

Color of button border and the right side sign (ellipsis or triangle)[369] when a button is under the mouse pointer.

```
property HoverLineColor: TColor;
```

Default value:

clInfoText

See also:

- Color[369];
- HoverColor[370];
- HoverColor[370].

Index of a button image in ImageList[370].

```
property ImageIndex: Integer;
```

This image is displayed on a button.

You can change this property in TCustomRichViewEdit.OnCaretMove[356] (to create the proper button for the item at the position of caret)

Default value:

0

Link to an image list containing images for buttons.

```
property ImageList: TCustomImageList;
```

The ImageIndex[370]-th image is shown on button.

You can change this property in TCustomRichViewEdit.OnCaretMove[356] (to create the proper button for the item at the position of caret).

Border color of a button.

```
property LineColor: TColor;
```

Default value:

clHighlight

See also:

- Color[369];
- HoverColor[370];
- HoverLineColor[370].

Link to popup menu to display when the user clicks a button or presses ShortCut[371].

```
property Menu: TPopupMenu;
```

Alternatively, you can process TCustomRichViewEdit.OnSmartPopupClick[366] event to execute a command or display a dialog.

You can change this property in TCustomRichViewEdit.OnCaretMove[356] (to create the proper button for the item at the position of caret).

Defines the position of a smart popup button relative to the current item.

```
type
  TRVSmartPopupPosition =
    (rvsppTopLeft, rvsppTopRight, rvsppBottomRight, rvsppBottomLeft);
property Position: TRVSmartPopupPosition;
```

Note: TRVSmartPopupPosition is defined in RVScroll unit.

Specifies the key combination users can type to execute command associated with visible "smart popup" button.

```
property ShortCut: TShortCut;
```

The effect of pressing ShortCut is the same as clicking the button.

Default value:

$6028 (Shift + Ctrl + ⇩)

See also properties:

- Menu[371].

See also events of TCustomRichViewEdit:

- OnSmartPopupClick[366].

Call SetButtonState(*True*) to fix the button in the "hot" state. It's useful while you display your own drop down window for the button. Call SetButtonState(*False*) to return it back to normal.

```
procedure SetButtonState(Hot: Boolean);
```

See also events of TCustomRichViewEdit:

- OnSmartPopupClick[366].

1.4.1.2.5 Examples

1.4.1.2.5.1 Moving caret to the beginning of paragraph

This example shows how to move caret to the beginning of the ParagraphIndex-th paragraph

```
procedure GoToParagraph(rve: TCustomRichViewEdit[282]; ParagraphIndex: Integer);
var i: Integer;
begin
  for i := 0 to rve.ItemCount[144]-1 do begin
    if rve.IsParaStart[208](i) then
      dec(ParagraphIndex);
    if ParagraphIndex<0 then begin
      rve.SetSelectionBounds[235](i, rve.GetOffsBeforeItem[201](i),
        i, rve.GetOffsBeforeItem[201](i));
      rve.Invalidate;
      exit;
    end;
  end;
```

```
end;
```

Call:

```
GoToParagraph(RichViewEdit1, 7);
```

ParagraphIndex is zero-based.

1.4.1.2.5.2 OnDropFile Example

By default, TRichViewEdit accepts the following types of dropped files:

- graphics,
- RVF,
- RTF,
- plain text (ANSI).

This example shows how to use OnDropFiles[359] event to insert all these types of files, and all file formats supported by installed Microsoft Office text import converters.

This example assumes that RVOfficeConverter1:TRVOfficeConverter[496] is placed on the form.

```
procedure TForm1.RichViewEdit1DropFiles(Sender: TCustomRichViewEdit;
  Files: TStrings; var FileAction: TRVDropFileAction;
var DoDefault: Boolean);
  {.....................................................}
  { This function inserts one file FileName }
  function InsertFile(const FileName: String): Boolean;
  var pic: TPicture;
  gr: TGraphic;
  Ext: String;
  i: Integer;
  begin
    Result := False;
    try
      // 1. Trying to insert as a graphic
      pic := TPicture.Create;
      try
        pic.LoadFromFile(FileName);
        gr := RV_CreateGraphics[672](TGraphicClass(pic.Graphic.ClassType));
        gr.Assign(pic.Graphic);
        Sender.InsertPicture[322]('', gr, rvvaBaseline[648]);
        Result := True;
      except;
      end;
      pic.Free;
      if Result then
        exit;
      // 2. Trying to insert as RTF file
      Ext := LowerCase(ExtractFileExt(FileName));
      if Ext='.rtf' then begin
        Sender.InsertRTFFromFileEd[323](FileName);
        Result := True;
        exit;
      end;
      // 3. Trying to insert as RVF file
      if Ext='.rvf' then begin
        Sender.InsertRVFFromFileEd[324](FileName);
        Result := True;
        exit;
      end;
```

```
      // 4. Trying to insert as text file
      if Ext='.txt' then begin
        Sender.InsertTextFromFile(328)(FileName);
        Result := True;
        exit;
      end;
      // 5. Trying to insert using office converters
      for i := 0 to RVOfficeConverter1.ImportConverters(499).Count-1 do
        if Pos(Ext, RVOfficeConverter1.ImportConverters[i].Filter)>0 then
          if RVOfficeConverter1.ImportRTF(501)(FileName, i) then begin
            RVOfficeConverter1.Stream(500).Position := 0;
            Sender.InsertRTFFromFileEd(323)(RVOfficeConverter1.Stream);
            RVOfficeConverter1.Stream.SetSize(0);
            Result := True;
            exit;
          end;
    except
    end;
  end;
  {.....................................................}
var i: Integer;
begin
  for i := 0 to Files.Count-1 do
    InsertFile(Files[i]);
  DoDefault := False;
end;
```

1.4.1.3 TDBRichView

Use TDBRichView to display hypertext documents stored in database field.

Unit DBRV;

Syntax

```
TDBRichView = class(TCustomRichView)
```

DBRichView does not display its content if it is not linked to RVStyle component via Style(156) property.

The most important property settings can be done in the component editor(134) for TRichView. In Delphi (C++Builder) IDE, right click the DBRichView object on the form, choose "Settings" in the context menu.

If the application doesn't require the data-aware capabilities of TDBRichView, use TRichView(128) instead, to conserve system resources.

For editing documents in database fields, use TDBRichViewEdit(377).

Hierarchy

TObject
TPersistent
TComponent
TControl
TWinControl
TCustomControl
TRVScroller(508)
TCustomRichView(128)

Field Type

This component must be linked to BLOB field.

If you use RVF [59] (RichView Format), the database field must have binary BLOB type (i.e. must have ability to store arbitrary binary data, without conversion and filtering of characters). For example, in Paradox Tables it is "Binary", in Microsoft Access tables it is "OLE Object".

You can also use text BLOB field type (memo). Memo fields cannot store binary data, so it is impossible to store documents in RVF format in them (unless you exclude *rvfoSaveBinary* from RVFOptions [152]).

However, you can use memo fields:

- to display text or RTF (Rich Text Format) fields,
- to display text mode RVF (that was saved with *rvfoSaveBinary* excluded from RVFOptions [152]).

Starting from Delphi 2006, Unicode memo fields (Field [375].DataType=*ftWideMemo*) are supported specially. TDBRichView does not attempt loading RVF from these fields. It tries to load Unicode RTF (RTF code converted from String to WideString) or Unicode text from them.

If *rvfoLoadBack* is in RVFOptions [152], the component clears BackgroundBitmap [136] before loading data from the field.

Supported Data Formats

The supported field formats are:

- RVF [59] (RichView Format),
- RTF (Rich Text Format),
- text,
- others (using events).

For all field types except for Unicode memos:

1. OnLoadCustomFormat [376] event occurs.
2. If this event is not processed, TDBRichView tries to load RVF.
3. If the field does not contain RVF, it tries to load RTF.
4. If the field does not contain RTF, it loads plain ANSI text.

For Unicode memos:

1. OnLoadCustomFormat [376] event occurs.
2. If this event is not processed, TDBRichView tries to load Unicode RTF (RTF code stored as WideString).
3. If the field does not contain RTF, it loads a plain Unicode text.

Events

OnNewDocument [377] event occurs when creating a new document or before loading an existing one from the field.

When the document is loaded from the field, OnLoadDocument [376] occurs.

OnLoadCustomFormat [376] allows loading documents in custom formats.

1.4.1.3.1 Properties

1.4.1.3.1.1 TDBRichView .AutoDisplay

Determines whether to automatically display the contents of field in the TDBRichView control.

```
property AutoDisplay: Boolean;
```

If AutoDisplay is *True*, the TDBRichView control automatically displays new data when the underlying BLOB field changes (such as when moving to a new record).

If AutoDisplay is *False*, the control shows only the field name whenever the underlying BLOB field changes. To display the data, the user can double-click on the control.

The effect of AutoDisplay is not purely cosmetic. When AutoDisplay is *False*, if the data changes, the content of TDBRichView document changes to the name of the field (written in the 0th text and paragraph style). Thus, if

AutoDisplay is *False*, applications should be cautious about using the methods for saving contents to ascertain the value of the underlying field.

Calling the LoadField 376 method causes component to update to the current value of the BLOB field. This change will also be reflected in the appearance of the control on screen.

Change the value of AutoDisplay to *False* if the automatic loading of BLOB fields seems to take too long.

1.4.1.3.1.2 TDBRichView.DataField

Specifies the field from which the TDBRichView control displays data.

```
property DataField: String;
```

Use DataField to bind the TDBRichView control to the field in the dataset. To fully specify database field, both the dataset and the field within that dataset must be defined. The DataSource 375 property of the TDBRichView control specifies the dataset which contains the DataField.

Please use only BLOB fields with TDBRichView.

1.4.1.3.1.3 TDBRichView.DataSource

Links the TDBRichView control to the dataset that contains the field it represents.

```
property DataSource : TDataSource;
```

Use DataSource to link the TDBRichView control to the dataset in which the data can be found. To fully specify database field for the DBRichView control, both the dataset and the field within that dataset must be defined. Use the DataField 375 property to specify the particular field within the dataset.

1.4.1.3.1.4 TDBRichView.Field

Indicates the TField object for the database field the TDBRichView control represents.

```
property Field : TField;
```

Read Field to get direct access to the content and properties of the database field.

If you use RVF 59 (RichView Format), the database field must have binary BLOB type (i.e. must have ability to store arbitrary binary data, without conversion and filtering of characters).

Using memo is not recommended (and impossible if you want to save styles, controls, pictures and tables in binary mode, or use Unicode).

However, you can use memo fields:

- to display text or RTF (Rich Text Format) fields,
- to display text mode RVF (that was saved with *rvfoSaveBinary* excluded from RVFOptions 152).

1.4.1.3.2 Methods

1.4.1.3.2.1 TDBRichView.Create

A constructor, creates new DBRichView

```
constructor Create(AOwner: TComponent);override;
```

Use Create to instantiate DBRichView at runtime. DBRichViews placed on forms at design time are created automatically. Specify the owner of the new DBRichView using the **AOwner** parameter.

Important note: initial values of several properties of components created in code may be different from initial values of components placed on form at design time. In the latter case, initial values of these properties can be overridden by the designtime component editor 134 (and overridden by default!)

1.4.1.3.2.2 TDBRichView .Destroy

A destructor, frees an instance of a DBRichView object.

```
destructor Destroy;override;
```

Destroy seldom needs to be called. When DBRichView is constructed without an owner by the Create 375 method, Free should be called to release memory and dispose of the object. Free checks to see if the pointer is *nil* before calling Destroy.

1.4.1.3.2.3 TDBRichView .LoadField

LoadField loads the document from the field in the database into the TDBRichView control.

```
procedure LoadField;
```

LoadField allows an application to control when the TDBRichView control displays the contents of the BLOB field. Use LoadField to update the document in the TDBRichView control to the value of the field in the current record. LoadField only works when the AutoDisplay 374 property is *False*, so that the value of the field is not loaded automatically.

Similar to LoadMemo in TDBRichEdit or TDBMemo.

1.4.1.3.3 Events

1.4.1.3.3.1 TDBRichView .OnLoadCustomFormat

This event allows loading data from field in your own format.

```
type
  TRVCustomFormatEvent = procedure (Sender: TCustomRichView 128;
    Stream: TStream; var DoDefault: Boolean) of object;
property OnLoadCustomFormat: TRVCustomFormatEvent;
```

(introduced in version 10)

Input parameters:

Stream contains data from DataField 375. Data should be loaded from **Stream** to **Sender**.

DoDefault is *True*.

Output parameters:

If data are loaded successfully, set **DoDefault** to *False*, otherwise the component will attempt to load RVF/RTF/ text.

1.4.1.3.3.2 TDBRichView .OnLoadDocument

Occurs after loading document from the database field

```
property OnLoadDocument: TNotifyEvent;
```

(introduced in v1.9)

This event allows to preprocess the loaded document before displaying.

See also:

- OnNewDocument 377.

1.4.1.3.3.3 TDBRichView .OnNew Document

Occurs when creating or before loading document

```
property OnNewDocument: TNotifyEvent;
```

(introduced in v1.9)

Occurs:

- on creating new document (before displaying field from newly added record);
- before loading document from the field.

Use this event to assign default values to properties (such as background, margins, styles).

See also:

- OnLoadDocument 376.

1.4.1.4 TDBRichViewEdit

Use TDBRichView to edit hypertext documents stored in database field.

Unit DBRV;

Syntax

```
TDBRichViewEdit = class(TCustomRichViewEdit)
```

DBRichViewEdit does not display or edit its content if it is not linked to RVStyle component via Style 156 property.

The most important property settings can be done in the component editor 134 for TRichView. In Delphi (C++Builder) IDE, right click the DBRichViewEdit object on the form, choose "Settings" in the context menu. If you decide to choose "Allow adding styles dynamically" mode, please set AutoDeleteUnusedStyles 378 to *True*.

If the application doesn't require the data-aware capabilities of TDBRichViewEdit, use TRichViewEdit 282 instead, to conserve system resources.

If the application doesn't require the editing capabilities of TDBRichViewEdit, use TDBRichView 373 instead, to conserve system resources.

Hierarchy

TObject
TPersistent
TComponent
TControl
TWinControl
TCustomControl
TRVScroller 508
TCustomRichView 128
TCustomRichViewEdit 282

Field Type

This component must be linked to BLOB field.

If you use RVF 59 (RichView Format), the database field must have binary BLOB type (i.e. must have ability to store arbitrary binary data, without conversion and filtering of characters). For example, in Paradox Tables it is "Binary", in Microsoft Access tables it is "OLE Object".

You can also use text BLOB field type (memo). Memo fields cannot store binary data, so it is impossible to store documents in RVF format in them (unless you exclude *rvfoSaveBinary* from RVFOptions 152).

However, you can use memo fields:

- to display text or RTF (Rich Text Format) fields,
- to display text mode RVF (that was saved with *rvfoSaveBinary* excluded from RVFOptions[152]).

Starting from Delphi 2006, Unicode memo fields (Field[379].DataType=*ftWideMemo*) are supported specially. TDBRichView does not attempt loading RVF from these fields. It tries to load Unicode RTF (RTF code converted from String to WideString) or Unicode text from them.

If *rvfoLoadBack* is in RVFOptions[152] and FieldFormat[380]=*rvdbRVF*, the component clears BackgroundBitmap[136] before loading data from the field.

Supported Data Formats

The supported field formats are:

- RVF[59] (RichView Format),
- RTF (Rich Text Format),
- text,
- others (using events).

For all field types except for Unicode memos:

1. OnLoadCustomFormat[381] event occurs.
2. If this event is not processed, TDBRichView tries to load RVF.
3. If the field does not contain RVF, it tries to load RTF.
4. If the field does not contain RTF, it loads plain ANSI text.

For Unicode memos:

1. OnLoadCustomFormat[381] event occurs.
2. If this event is not processed, TDBRichView tries to load Unicode RTF (RTF code stored as WideString).
3. If the field does not contain RTF, it loads plain Unicode text.

Saving data to the field:

OnSaveCustomFormat[382] event allows saving data in your own format. If this event is not processed, documents are saved in FieldFormat[380].

Events

OnNewDocument[382] event occurs when creating a new document or before loading an existing one from the field.

When the document is loaded from the field, OnLoadDocument[382] occurs.

OnLoadCustomFormat[381] and OnSaveCustomFormat[382] allow saving and loading documents in custom formats.

Examples

Example 1: Using viewer-style methods in DBRichViewEdit[383] (such as LoadRTF[209])

Example 2: Inserting controls what can modify themselves in DBRichViewEdit[383] (for example, inserting TEdit; user can modify its text)

1.4.1.4.1 Properties

1.4.1.4.1.1 TDBRichViewEdit.AutoDeleteUnusedStyles

If this property is set to *True*, the editor deletes unused text[404], paragraph[398] and list[396] styles before posting to the database field.

```
property TDBRichViewEdit.AutoDeleteUnusedStyles: Boolean;
```

(introduced in version 1.7)

Assign *True* to this property only if:

- this editor uses TRVStyle[385] object exclusively (no other RichViews are linked to this RVStyle via Style[156]

property) and
- collections of text, paragraph and list styles are saved together with documents in the RVF field (see RVFOptions[152], *rvfoSaveTextStyles* and *rvfoSaveParaStyles* must be included)

If FieldFormat[380] is a plain text, this property is ignored,

Default value:

False

1.4.1.4.1.2 TDBRichView Edit.AutoDisplay

Determines whether to automatically display the contents of field in the TDBRichViewEdit control.

```
property AutoDisplay: Boolean;
```

If AutoDisplay is *True*, the TDBRichViewEdit control automatically displays new data when the underlying BLOB field changes (such as when moving to a new record).

If AutoDisplay is *False*, the control shows only the field name whenever the underlying BLOB field changes. To display the data, the user can double-click on the control.

The effect of AutoDisplay is not purely cosmetic. When AutoDisplay is *False*, if the data changes, the content of TDBRichViewEdit document changes to the name of the field (written in the 0th text and paragraph style). Thus, if AutoDisplay is *False*, applications should be cautious about using the methods for saving contents to ascertain the value of the underlying field.

Calling the LoadField[381] method causes component to update to the current value of the BLOB field. This change will also be reflected in the appearance of the control on screen.

Change the value of AutoDisplay to *False* if the automatic loading of BLOB fields seems to take too long.

1.4.1.4.1.3 TDBRichView Edit.DataField

Specifies the field from which the TDBRichViewEdit control displays data.

```
property DataField: String;
```

Use DataField to bind the TDBRichViewEdit control to the field in the dataset. To fully specify database field, both the dataset and the field within that dataset must be defined. The DataSource[379] property of the TDBRichView control specifies the dataset which contains the DataField.

Please use only BLOB fields with TDBRichViewEdit.

1.4.1.4.1.4 TDBRichView Edit.DataSource

Links the TDBRichViewEdit control to the dataset that contains the field it represents.

```
property DataSource : TDataSource;
```

Use DataSource to link the TDBRichViewEdit control to the dataset in which the data can be found. To fully specify database field for the DBRichViewEdit control, both the dataset and the field within that dataset must be defined. Use the DataField[379] property to specify the particular field within the dataset.

1.4.1.4.1.5 TDBRichView Edit.Field

Indicates the TField object for the database field the TDBRichViewEdit control represents.

```
property Field : TField;
```

Read Field to get direct access to the content and properties of the database field.

If you use RVF[59] (RichView Format), the database field must have binary BLOB type (i.e. must have ability to store arbitrary binary data, without conversion and filtering of characters).

Using memo is not recommended (and impossible if you want to save styles, controls, pictures and tables in binary mode, or use Unicode).

However, you can use memo fields:

- to display text or RTF (Rich Text Format) fields;
- to display text mode RVF (that was saved with *rvfoSaveBinary* excluded from RVFOptions 152).

1.4.1.4.1.6 TDBRichView Edit.FieldFormat

Format for saving data in the database field

```
property FieldFormat: TRVDBFieldFormat;

type
  TRVDBFieldFormat = (rvdbRVF, rvdbRTF, rvdbText);
```

(introduced in version 1.6)

This property allows changing the field saving format to RTF or plain text.

Note: It is not guaranteed that document saved in RTF and text will be the same after reloading (some item types, text and paragraph attributes cannot be saved in RTF or plain text).

You can save data in your own format using OnSaveCustomFormat 382 event.

Starting from Delphi 2006, Unicode memo fields (Field 379.DataType=*ftWideMemo*) are supported specially. For these fields, if FieldFormat is *rvdbRVF* or *rvdbRTF*, documents are stored as Unicode RTF (RTF code converted from String to WideString). If FieldFormat is *rvdbText*, documents are stored as Unicode text.

Default value:

rvdbRVF

1.4.1.4.1.7 TDBRichView Edit.IgnoreEscape

Disallows processing of Escape key

```
property IgnoreEscape: Boolean;
```

(introduced in version 1.8)

By default, Escape cancels editing in TDBRichViewEdit.

If you set this property to *True*, Escape will do nothing.

Default value:

False

1.4.1.4.1.8 TDBRichView Edit.ReadOnly

Determines whether the user can use the DBRichVewEdit control to change the value of the field in the current record.

```
property ReadOnly : Boolean;
```

Set ReadOnly to *True* to prevent the user from changing the content of the database field. When ReadOnly is *True*, the edit control is used only to display the content of the field. If ReadOnly is *False*, the user can change the field's value as long as the dataset is in edit mode.

Note: TRichViewEdit.ReadOnly 294 property is overridden in TDBRichViewEdit to get/set read-only mode in the linked dataset.

1.4.1.4.2 Methods

1.4.1.4.2.1 TDBRichViewEdit.Change

Generates OnChange[357] event.

```
procedure Change; override;
```

TCustomRichViewEdit.Change[303] method was overridden to support data-aware features of TDBRichViewEdit.

In addition to the actions inherited from RichViewEdit, this method also informs the linked dataset that data were modified.

Change can be useful if you need to modify document using viewer-style methods, see the example: Using viewer-style methods in DBRichViewEdit[383].

1.4.1.4.2.2 TDBRichViewEdit.Create

A constructor, creates new DBRichViewEdit.

```
constructor Create(AOwner: TComponent);override;
```

Use Create to instantiate DBRichViewEdit at runtime. DBRichViewEdits placed on forms at design time are created automatically. Specify the owner of the new DBRichViewEdit using the **AOwner** parameter.

Important note: initial values of several properties of components created in code may be different from initial values of components placed on form at design time. In the latter case, initial values of these properties can be overridden by the designtime component editor[134] (and overridden by default!)

1.4.1.4.2.3 TDBRichViewEdit.Destroy

A destructor, frees an instance of a DBRichViewEdit object

```
destructor Destroy;override;
```

Destroy seldom needs to be called. When DBRichViewEdit is constructed without an owner by the Create[381] method, Free should be called to release memory and dispose of the object. Free checks to see if the pointer is *nil* before calling Destroy.

1.4.1.4.2.4 TDBRichViewEdit.LoadField

LoadField loads the document from the field in the database into the TDBRichViewEdit control.

```
procedure LoadField;
```

LoadField allows an application to control when the TDBRichViewEdit control displays the contents of the BLOB field. Use LoadField to update the document in the TDBRichViewEdit control to the value of the field in the current record. LoadField only works when the AutoDisplay[379] property is *False*, so that the value of the field is not loaded automatically.

Similar to LoadMemo in TDBRichEdit or TDBMemo.

1.4.1.4.3 Events

1.4.1.4.3.1 TDBRichViewEdit.OnLoadCustomFormat

This event allows loading data from field in your own format.

```
type
  TRVCustomFormatEvent = procedure (Sender: TCustomRichView[128];
    Stream: TStream; var DoDefault: Boolean) of object;
property OnLoadCustomFormat: TRVCustomFormatEvent;
```

(introduced in version 10)

Input parameters:

Stream contains data from DataField[379]. Data should be loaded from **Stream** to **Sender**.

DoDefault is *True*.

Output parameters:

If data are loaded successfully, set **DoDefault** to *False*, otherwise the component will attempt to load RVF/RTF/text.

See also events:

- OnSaveCustomFormat[382].

1.4.1.4.3.2 TDBRichView Edit.OnLoadDocument

Occurs after loading document from the database field

```
property OnLoadDocument: TNotifyEvent;
```

(introduced in v1.9)

This event allows to preprocess the loaded document before displaying

See also:

- OnNewDocument[382].

1.4.1.4.3.3 TDBRichView Edit.OnNew Document

Occurs when creating or before loading document

```
property OnNewDocument: TNotifyEvent;
```

(introduced in v1.9)

Occurs:

- on creating new document (before displaying field from newly added record);
- before loading document from the field.

Use this event to assign default values to properties (such as background, margins, styles).

See also:

- OnLoadDocument[382].

1.4.1.4.3.4 TDBRichView Edit.OnSaveCustomFormat

This event allows saving data to field in your own format.

```
type
  TRVCustomFormatEvent = procedure (Sender: TCustomRichView[128];
    Stream: TStream; var DoDefault: Boolean) of object;
property OnSaveCustomFormat: TRVCustomFormatEvent;
```

(introduced in version 10)

Stream is empty initially. Save the content of **Sender** in Stream and set **DoDefault** to *False*, otherwise the component will save document according to the FieldFormat[380] property.

See also events:

- OnLoadCustomFormat[381].

1.4.1.4.4 Examples

1.4.1.4.4.1 Using viewer-style methods in DBRichViewEdit

This example shows how to use viewer-style methods in DBRichViewEdit (i.e. methods, introduced in TCustomRichView[128]).

DBRichViewEdit1 edits a field of table Table1.We want to load RTF file.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if OpenDialog1.Execute and DBRichViewEdit1.CanChange[302] then begin
    DBRichViewEdit1.Clear[304];
    DBRichViewEdit1.LoadRTF[209](OpenDialog1.FileName);
    DBRichViewEdit1.Change[381];
    DBRichViewEdit1.Format[182];
  end;
end;
```

Calling CanChange has the same effect as calling Table1.Edit.

If you want to save changes in the database immediately, call Table1.Post instead of DBRichViewEdit1.Format.

This sequence of actions (CanChange - modifications - Change - Format) is required only for viewer-style methods.

Editing-style methods do this work automatically:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  if OpenDialog1.Execute then
    DBRichViewEdit1.InsertRTFFromFileEd[323](OpenDialog1.FileName);
end;
```

1.4.1.4.4.2 Inserting controls what can modify themselves in DBRichViewEdit

This example shows how to work with TEdit inserted in TDBRichViewEdit[377], if you want to store its text in RVF field.

The problem: when user changes text in TEdit, the dataset is not informed that the document is changed, and these changes are not saved.

In this example, we use DBRichViewEdit1:TDBRichViewEdit placed on Form1.

1) Add in the private section of Form1:

```
procedure EditChange(Sender: TObject);
```

Implementation:

```
procedure TForm1.EditChange(Sender: TObject);
begin
  if DBRichViewEdit1.CanChange[302] then
    DBRichViewEdit1.Change[381]
  else
    Beep;
end;
```

2) When creating a new TEdit, assign its OnChange event:

```
var Edit: TEdit;
begin
  Edit := TEdit.Create(nil);
  Edit.OnChange := EditChange;
  DBRichViewEdit1.InsertControl[318]('', Edit, rvvaBaseline);
end;
```

3) You'll need to assign this event to all edits loaded from the database field with RichView documents. Use OnControlAction[239] of DBRichViewEdit:

```
procedure TForm1.DBRichViewEdit1ControlAction(Sender: TCustomRichView;
  ControlAction: TRVControlAction; ItemNo: Integer; var ctrl: TControl);
begin
  if (ControlAction=rvcaAfterRVFLoad) and (ctrl is TEdit) then
    TEdit(ctrl).OnChange := EditChange;
end;
```

And of course, do not forget to register TEdit in the initialization section of unit:

```
initialization
  RegisterClasses([TEdit]);
```

1.4.1.5 TRVPrintPreview

This component displays a print preview for TRichView[128], TRichViewEdit[282], TDBRichView[373], or TDBRichViewEdit[377].

TRVPrintPreview control can display preview only if it is linked to RVPrint[384], and RVPrint is ready (it has a document assigned and pages formatted)

Unit RVPP;

Syntax

```
TRVPrintPreview = class(TCustomRVPrintPreview[520])
```

Hierarchy

TObject
TPersistent
TComponent
TControl
TWinControl
TCustomControl
TRVScroller[508]
TCustomRVPrintPreview[520]

Information

See more info in the topic about the ancestor of TRVPrintPreview, TCustomRVPrintPreview[520].

The main property of this component is RVPrint[384], allowing to link it with TRVPrint[479] object.

1.4.1.5.1 Properties

1.4.1.5.1.1 TRVPrintPreview.RVPrint

This property links this preview control to  TRVPrint component.

```
property RVPrint: TRVPrint[479];
```

TRVPrintPreview control can display preview only if it is linked to RVPrint, and RVPrint is ready (it has a document assigned and pages formatted).

1.4.2 Additional Components

1.4.2.1 TRVStyle

This invisible at run-time component is used for defining a visual appearance of RichView[128] and its descendant components.

Unit RVStyle

Syntax

```
TRVStyle = class(TComponent)
```

Hierarchy

TObject
TPersistent
TComponent

Using

If you want to display some document in RichView, associate this RichView with some RVStyle component: create TRVStyle component and assign it to the Style[156] property of RichView. You can do it at design-time using the the Object Inspector.

The same RVStyle object can be used by several RichView[128], RichViewEdit[282], DBRichView[373], DBRichViewEdit[377] controls.

But if you use the following:

- TCustomRichView.DeleteUnusedStyles[181];
- TDBRichViewEdit.AutoDeleteUnusedStyles[378]=*True*;
- TCustomRichView.LoadRVF[211], LoadRVFFromStream[210], and RVFTextStylesReadMode[154] or RVFParaStylesReadMode[154] <> *rvf_sIgnore*;
- TDBRichView or TDBRichViewEdit, and RVFTextStylesReadMode[154] or RVFParaStylesReadMode[154] <> *rvf_sIgnore*

you must use one RVStyle for one RichView.

Design-Time Component Editor

TRVStyle has a design-time component editor (unit RVSEdit). This editor:

- adds 3 items into the context menu for component:"Edit Text Styles", "Edit Paragraph Styles", "Edit List Styles" which invoke property editors for TextStyles[404], ParaStyles[398], or ListStyles[396];
- adds "Convert to twips" or "Convert to pixels" item in the context menu, allowing to convert all RVStyle properties measured in Units[404].
- adds 3 items into the context menu for component:"Edit Style Templates", "Save Style Templates...", "Load Style Templates..." which invoke property editors for StyleTemplates[402], save them to a file, and load them from a file.
- invokes property editor for TextStyles on double-click.

Properties

Three main properties of RVStyle are:

- TextStyles[404] – collection of text attributes (styles), a collection of TFontInfo[444] items;
- ParaStyles[398] – collection of paragraph attributes (styles), a collection of TParaInfo[456] items;
- ListStyles[396] – collection of paragraph list styles (bullets and numbering), a collection of TRVListInfo[459] items.

Style templates ("real" styles)

- StyleTemplates[402] – collection of "real" styles, controlling properties of TextStyles[404] and ParaStyles[398];
- MainRVStyle[397] allows using a single StyleTemplates[402] in multiple RVStyle components.

Units of measurement

- Units[404] defines measure units for properties of this component and documents in linked controls.

RVStyle allows to set some colors for associated controls:

- Color[387] – background color;
- SelColor, InactiveSelColor[393] – background colors of selected text (with and without focus);
- SelTextColor, InactiveSelTextColor[393] – colors of selected text (with and without focus);
- DisabledFontColor[389] – text color for disabled controls.
- HoverColor[393] – default color of hypertext links under the mouse pointer;
- CheckpointColor, CheckpointEvColor[387] – colors of *checkpoints* (if visible);
- PageBreakColor[398] – color of explicit page breaks (if visible);
- FloatingLineColor[390] – color of placeholders for left- and right-aligned items (if visible).
- GridColor, GridReadOnlyColor[392] – colors of table grid lines (if visible); GridStyle, GridReadOnlyStyle[392] define their pen styles.

RVStyle allows to set cursors:

JumpCursor[394] – hypertext cursor for *hotspots*[97] and *hot-pictures*[93] (and default cursor for text hypertext items);
LineSelectCursor[394] – cursor for the left margin.

Tabs:

- SpacesInTab[402] – If positive, **Add***** methods of RichView, methods for loading text and RTF, keyboard input will replace tab characters with the specified number of spaces. If 0 (default), a special tabulator item type[89] is used;
- DefTabWidth[388] – default distance between tab stops.

Properties related to label items[101] and item types inherited from them:

- FieldHighlightColor[390] – color for highlighting label items.
- FieldHighlightType[390] specifies when to highlight label items.
- FootnoteNumbering[389] – numbering type for footnotes.
- FootnotePageReset[392] – numbering mode for footnotes.
- EndnoteNumbering[389] – numbering type for endnotes.
- SidenoteNumbering[389] – numbering type for sidenotes.

Properties related to selection:

- SelectionStyle[399] – visual appearance of selection;
- SelectionMode[399] – modes of selection;
- SelectionHandleKind[398] – visual appearance of touch screen selection handles;
- SelColor, InactiveSelColor[393] – background colors of selected text (with and without focus);
- SelTextColor, InactiveSelTextColor[393] – colors of selected text (with and without focus).

Other properties

- TextEngine[403] specifies which API is used for drawing and measuring text;
- FontQuality[391] – font quality and anti-aliasing;
- DefUnicodeStyle[389] – default Unicode text style;
- DefCodePage[388] – default code page for internal ANSI <->Unicode conversion;
- LineWrapMode[394] specifies a line wrapping algorithm;
- UseSound[404].

Methods

You can save and load all information in RVStyle in ini-file:

- SaveINI[409] saves RVStyle properties in ini-file;
- LoadINI[407] loads RVStyle properties from ini-file;

or registry:

- SaveReg[409] saves RVStyle properties in ini-file;
- LoadReg[408] loads RVStyle properties from ini-file.

You can export styles in style sheet (for saving HTML files):

- SaveCSS[408];
- SaveCSSToStream[409].

You can search for a style with the specified properties:

- FindTextStyle, FindParaStyle 407

Events

Custom drawing:

- OnDrawCheckpoint 411 – drawing checkpoint;
- OnDrawPageBreak 413 – drawing page break;
- OnDrawParaBack 414 – drawing background of paragraph;
- OnDrawTextBack 416 – drawing background of text;
- OnApplyStyle 410 – applying text style to Canvas;
- OnApplyStyleColor 411 – applying text style color to Canvas;
- OnDrawStyleText 415 – drawing text of the given style;
- OnStyleHoverSensitive 417 – "should the component be repainted when user moves the mouse pointer over hyperlink of the given style?".

1.4.2.1.1 Properties

1.4.2.1.1.1 TRVStyle.CheckpointColor, CheckpointColor

A color of dotted lines showing positions of *checkpoints* in document.

```
property CheckpointColor: TColor;
property CheckpointEvColor: TColor;
```

If TRichView shows its *checkpoints* (*rvoShowCheckpoints* in TRichView.Options 146), they are displayed as dotted horizontal lines with a small circle at the top left corner of the associated item.

Normal *checkpoints* are shown using **CheckpointColor**. *Checkpoints* with "RaiseEvent" flag are shown using CheckpointEvColor.

You can draw *checkpoints* yourself in OnDrawCheckpoint 411 event.

TRichView must be linked with this TRVStyle (see TRichView.Style 156).

Default value:

CheckpointColor: clGreen
CheckpointEvColor: clLime

See also events:

- OnDrawCheckpoint 411.

See also properties of TCustomRichView:

- Options 146.

See also:

- *Checkpoints* 30 overview 30.

1.4.2.1.1.2 TRVStyle.Color

A default background color for TRichView component linked to this TRVStyle component.

```
property Color: TColor;
```

This property may be overridden with TRichView.Color 139 property.

TRichView must be linked with this TRVStyle (see TRichView.Style 156).

See also properties of TCustomRichView:

- Style 156;
- Color 139;
- BackgroundStyle 136;
- BackgroundBitmap 136.

1.4.2.1.1.3 TRVStyle.CurrentItemColor

A color for drawing frame around the current item in TRichViewEdit[282].

```
property CurrentItemColor: TColor;
```

(Introduced in v1.9)

Current item is an item at the position of caret. All RichViewEdit's **GetCurrent***** and **SetCurrent***** methods work with this item.

Set this property to *clNone* to disable this effect.

TRichViewEdit must be linked with this TRVStyle (see TRichView.Style[156]).

Default value:

clNone

1.4.2.1.1.4 TRVStyle.DefCodePage

A default code page used inside TRichView for automatic conversions of ANSI strings to Unicode and vice versa

```
property DefCodePage: TRVCodePage[625];
```

(Introduced in version 1.4)

This property defines the code page (language) which will be used for ANSI<->Unicode conversions.

Usually, if conversion includes only one TRichView text item, the code page for conversion is defined by the character set[434] of the text style. If the conversion is performed on a block of text consisting of several items, TRichView gets a code page for the conversion from this property.

Note: there is a special property editor for this property allowing to select code page from dropdown list.

Default value:

0 (Windows default code page, *CP_ACP*)

See also:

- Unicode in RichView[64].

1.4.2.1.1.5 TRVStyle.DefTabWidth

A default distance between tab stops

```
property DefTabWidth: TRVStyleLength[644];
```

(introduced in v1.9)

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units[404].

Tab stops are started from the LeftMargin[145] (or from the RightMargin[149] for right-to-left[69] paragraphs).

Note: tabulators[89] can be inserted from text/RTF files or keyboard only if value of SpacesInTab[402] property is 0 (otherwise tabs are inserted as several space characters).

Default value:

48

See also properties:

- ParaStyles[398].Tabs[455].

See also properties:

- Units of measurement in TRichView[74]

1.4.2.1.1.6 TRVStyle.DefUnicodeStyle

A default text style for Unicode operations.

```
property DefUnicodeStyle: Integer;
```

(Introduced in version 1.3)

If set to value >=0, RichViewEdit[282] will switch to this style before inserting Unicode text (if the current style is not Unicode[446]).

Examples of such operations: inserting Unicode text (may be from file), pasting Unicode text, inserting data from IME (Input Method Editor).

Warning: be very careful when using this property in applications that store collection of text styles in RVF documents. Value of this property is not stored in RVF! Generally, it not recommended to use this property in the default "allow adding styles dynamically" mode, but it may be useful in "use a predefined set of styles" mode, see TRichView component editor[134].

Default value:

-1

See also properties:

- TextStyles[404].

See also:

- Unicode in RichView[64].

1.4.2.1.1.7 TRVStyle.DisabledFontColor

Defines a font color to use in disabled TRichView[128] controls.

```
property DisabledFontColor: TColor;
```

(Introduced in version 13)

If this property = *clNone*, a disabled control uses the same text color as enabled control.

Default value:

clNone

1.4.2.1.1.8 TRVStyle.EndnoteNumbering

A numbering type for endnotes[103], footnotes[104], and sidenotes[105].

```
property EndnoteNumbering: TRVSeqType
property FootnoteNumbering: TRVSeqType
property SidenoteNumbering: TRVSeqType
```

(introduced in version 10 and 15)

See TRVSeqType[643] for possible values.

If you changed value of these properties while the document in the linked[156] TRichView[128] control is displayed, call RichView.RefreshSequences[214] and RichView.Format[182].

Default values:

- EndnoteNumbering: *rvseqLowerRoman*
- FootnoteNumbering: *rvseqDecimal*
- SidenoteNumbering: *rvseqLowerAlpha*

See also:

- FootnotePageReset[392].

1.4.2.1.1.9 TRVStyle.FieldHighlightColor

A color for highlighting labels [101], numbered sequences [102], endnotes [103], footnotes [104], references to parent footnotes and endnotes [108].

```
property FieldHighlightColor: TColor;
```

(introduced in version 10)

See FieldHighlightType [390] for details.

Default value:

clBtnFace

1.4.2.1.1.10 TRVStyle.FieldHighlightType

Specifies when FieldHighlightColor [390] must be used for highlighting labels [101], numbered sequences [102], endnotes [103], footnotes [104], references to parent footnotes and endnotes [108].

```
type
  TRVFieldHighlightType =
    (rvfhNever, rvfhCurrent, rvfhAlways);
property FieldHighlightType;
```

(introduced in version 10)

Type	Meaning
rvfhNever	Items are never highlighted.
rvfhCurrent	Item at the position of the caret is highlighted. Either the editor must have the input focus, or its ForceFieldHighlight [293] property must be *True*.
rvfhAlways	All items are always highlighted.

Default value:

rvfhCurrent

1.4.2.1.1.11 TRVStyle.FloatingLineColor

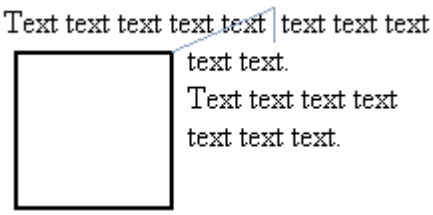
Color for displaying places of insertion for left- and right-aligned [648] items, and markers for text box items [107].

```
property FloatingLineColor: TColor;
```

(introduced in version 12)

Left- and right-aligned items

Two lines are drawn: a vertical line in place of the placeholder (where this object is inserted), and a line from the top of placeholder to the nearest corner of the floating object.



The lines are displayed in the following cases:

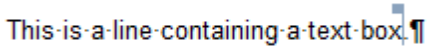
- in TCustomRichViewEdit [282], for one item, when this item is current (at the position of the caret)
- for all side-aligned items, if *rvoShowSpecialCharacters* is included in Options [146], and *rvcFloatingLines* is

included in RVVisibleSpecialCharacters 665

Assign *clNone* to prevent displaying these lines.

Text box items

A special mark is inserted in places of insertion of text box items 107:



The marks are displayed in the following cases:

- in TCustomRichViewEdit 282, for one item, when this item is current (at the position of the caret)
- for all text box items, if *rvoShowSpecialCharacters* is included in Options 146, and *rvcPlaceholders* is included in RVVisibleSpecialCharacters 665.

If FloatingLineColor=*clNone*, SpecialCharactersColor 402 is used. If SpecialCharactersColor=*clNone*, *clBtnShadow* is used.

ScaleRichView note: in TSRichViewEdit, this color is also used to draw lines to places of insertion of sidenotes 105 and text box items, and around the edited floating box. Assign *clNone* to prevent displaying these lines.

Default value:

$BDA28A

1.4.2.1.1.12 TRVStyle.FontQuality

Specifies the font quality.

```
type
  TFontQuality = (fqDefault, fqDraft, fqProof, fqNonAntialiased, fqAntialiased,
    fqClearType, fqClearTypeNatural);
property FontQuality: TFontQuality;
```

(introduced in version 15)

Type	Meaning
fqDefault	The quality is determined by the system settings.
fqDraft	For raster fonts, scaling is enabled; the font size can be increased but the quality may be lower. Also the font supports Bold, Italic, Underline or Strikeout if necessary. The quality is less important than when Proof is used.
fqProof	The quality of the characters is important, so for the raster fonts scaling is disabled and the font closest in size is chosen.
fqNonAntialiased	Fonts are never antialiased.
fqAntialiased	Fonts are always antialiased if they support it. The size of the font cannot be too small or too large.
fqClearType	Fonts are rendered using the ClearType antialiasing method.
fqClearTypeNatural	Fonts are rendered using the ClearTypeNatural antialiasing method.

Note: TFontQuality is defined in RVStyle.pas for Delphi versions prior to XE; for newer versions of Delphi, UITypes.TFontQuality is used instead.

Default value:

fqDefault

1.4.2.1.1.13 TRVStyle.FootnotePageReset

Specifies whether to reset numbering of footnotes [104] on each page.

```
property FootnotePageReset: Boolean;
```

(introduced in version 10)

This property affects only printing. Footnotes in TRichView are still numbered sequentially.

This property can be changed when RTF file is loaded, if RichView.RTFReadProperties [151].IgnoreNotes [271] =*False*.

Default value:

True

See also:

- FootnoteNumbering [389]

1.4.2.1.1.14 TRVStyle.FullRedraw

For internal use

```
property FullRedraw: Boolean;
```

This property is temporal.

If FullRedraw=*True*, TRichView is redrawn completely when user moves the mouse pointer to the hyperlink area that needs to be highlighted with hovercolor [436].

If FullRedraw=*False*, TRichView redraws only the highlighted part (with additional 5 pixels border).

TRichView must be linked with this TRVStyle (see TRichView.Style [156]).

Default value:

False

1.4.2.1.1.15 TRVStyle.GridColor, GridReadOnlyColor, GridStyle, GridReadOnlyStyle

Colors and styles of table [98] grid lines.

```
property GridColor: TColor;
property GridReadOnlyColor: TColor;
property GridStyle: TPenStyle;
property GridReadOnlyStyle: TPenStyle;
```

(introduced in version 14)

GridColor and GridStyle are used for drawing invisible borders of tables in editing mode.

GridReadOnlyColor and GridReadOnlyStyle are used for drawing invisible borders of tables in TRichView [128] or in TRichViewEdit [282] in read-only [294] mode.

Table borders can be invisible in the following cases:

- their width is 0 (see BorderWidth [534] and CellBorderWidth [535] properties);
- some sides are hidden (see VisibleBorders [544] and Cell [567].VisibleBorders [575] properties).

If a pen style is *psClear*, or a color is *clNone*, a grid is not shown. But the recommended way to hide grid lines in all tables in the component is excluding *rvoShowGridLines* from TRichView.Options [146].

You can hide/show table grid for the specific table, see *rvtoHideGridLines* in Table.Options [540] property.

TRichView must be linked with this TRVStyle (see TRichView.Style [156]).

Default values:

- GridColor: *clBtnFace;*
- GridReadOnlyColor: *clBtnFace;*
- GridStyle: *psDot;*

- GridReadOnlyStyle: *psClear.*

1.4.2.1.1.16 TRVStyle.HoverColor

Default text hover color.

```
property HoverColor: TColor;
```

Text hover color is a text color under the mouse pointer.

All hypertext styles have their own hover color (see TFontInfo.HoverColor[436]). If they are equal to *clNone*, this HoverColor is used.

Set to *clNone* to disable hover color effect.

TRichView must be linked with this TRVStyle (see TRichView.Style[156]).

Default value:

clNone

1.4.2.1.1.17 TRVStyle.InactiveSelColor, InactiveSelTextColor, SelColor, SelTextColor

Color of text and background of selected text in TRichView components.

```
property SelColor: TColor;
property SelTextColor: TColor;
property InactiveSelColor: TColor;
property InactiveSelTextColor: TColor;
```

("Inactive" properties are introduced in version 1.6)

SelColor and SelTextColor are used when the control is focused. InactiveSelColor and InactiveSelTextColor are used otherwise.

Background colors (SelColor and InactiveSelColor) are used for:

- drawing background for the selected text;
- shading the selected *bullets*[96] and *hotspots*[97],
- drawing a frame around the selected *breaks*[93],
- drawing background for the selected pictures[90] and controls[94],
- drawing background for selected table[98] cells.

Set InactiveSelColor and InactiveSelTextColor to *clNone* to hide a selection when the input focus shifts to another control.

TRichView must be linked with this TRVStyle (see TRichView.Style[156]).

Default values:

- SelColor: *clHighlight*
- SelTextColor: *clHighlightText*
- InactiveSelColor: *clHighlight*
- InactiveTextColor: *clHighlightText*

See also properties:

- SelectionStyle[399].

1.4.2.1.1.18 TRVStyle.InvalidPicture

Picture that will be inserted when reading RTF or RVF containing invalid or missing pictures.

```
property InvalidPicture: TPicture;
```

(Introduced in version 1.7)

Default value:

 (TBitmap)

See also properties of TCustomRichView:

- RVFWarnings [155].

1.4.2.1.1.19 TRVStyle.JumpCursor

Mouse cursor for hypertext indication.

```
property JumpCursor : TCursor
```

This cursor appears when user moves the mouse pointer to *hotspot* [97] or *hot-picture* [93] item. In older versions of TRichView this value was also used for text hypertext items, but now all text styles can have their own hypertext cursors (see TFontInfo.JumpCursor [438]).

TRichView must be linked with this TRVStyle (see TRichView.Style [156]).

Default value:

crJump (101) (system hypertext cursor or )

See also:

- Cursors used by the components of RichView family [77].

1.4.2.1.1.20 TRVStyle.LineSelectCursor

Mouse cursor for line selection

```
property LineSelectCursor : TCursor
```

(introduced in v1.8)

This cursor is displayed when the mouse pointer is above the left margin.

This cursor is used in the special text selection mode: line selection. Line selection is started if the user started selecting with the mouse from the left margin.

Line selection does not work for right-to-left documents [137].

TRichView must be linked with this TRVStyle (see TRichView.Style [156]).

You can disable line selection if you set this property to *crNone* (-1).

Default value:

 *crRVFlipArrow* (106)

See also properties of TCustomRichView:

- LeftMargin [145].

See also:

- Cursors used by the components of RichView family [77].

1.4.2.1.1.21 TRVStyle.LineWrapMode

Specifies how text lines are wrapped.

```
type
  TRVLineWrapMode = (rvWrapAnywhere, rvWrapSimple, rvWrapNormal);
property LineWrapMode: TRVLineWrapMode
```

(introduced in version 13)

Value	Line wrapping algorithm

rvWrapAnywhere	Line break can occur in any place of text
rvWrapSimple	Line breaks inside text items [88] are added using line breaking rules. Line breaks between text items [88] can be added between any text items.
rvWrapNormal	Line breaks both inside and between text items are added using line breaking rules.

Line breaking rules are different for ANSI and Unicode [64] text items.

For ANSI text, line breaks occur only on space characters.

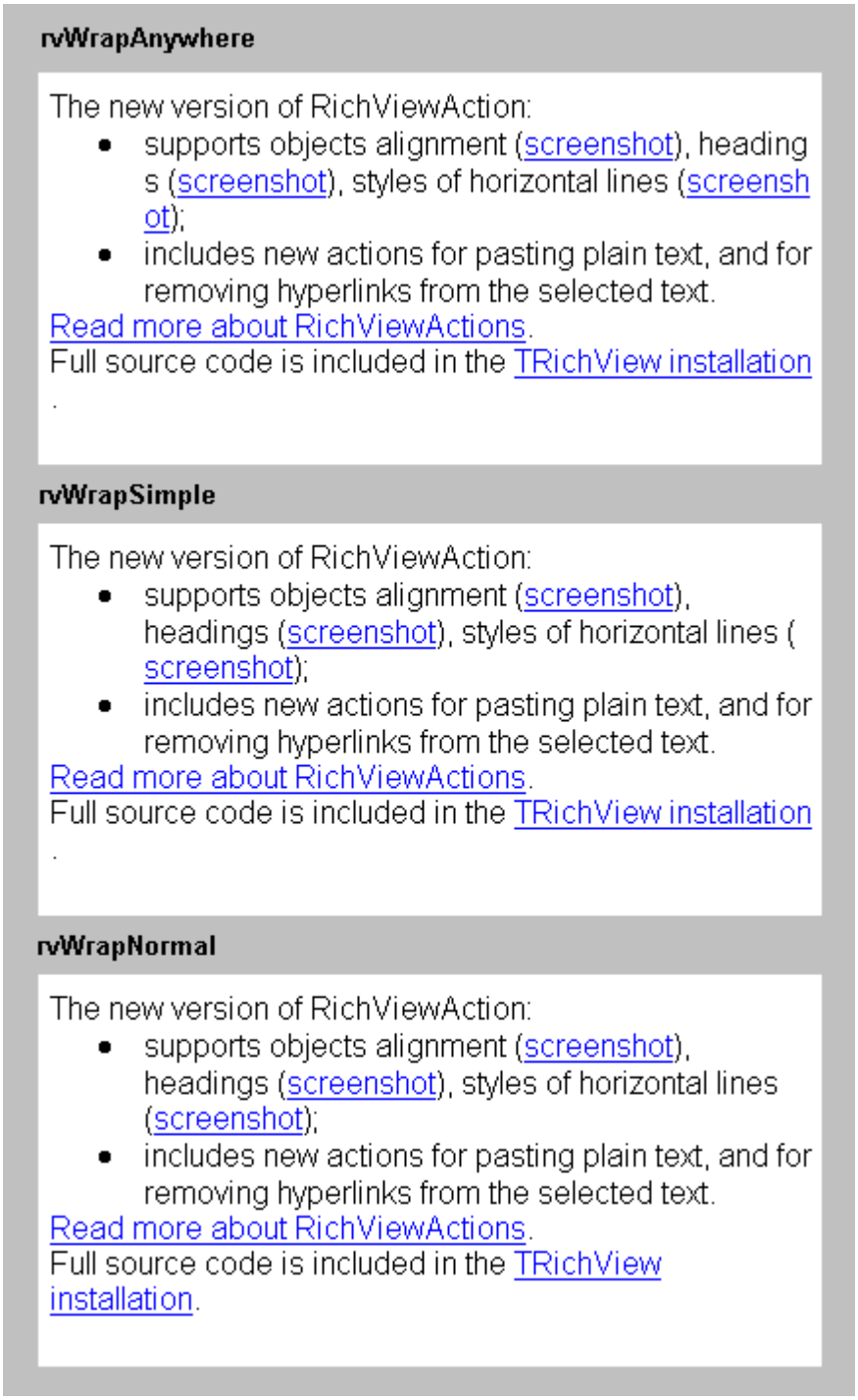
For Unicode text, line breaks are added using advanced rules. For example, in the string '(a)(b)' the only possible line break is between ')' and '('.

rvWrapAnywhere is a special mode.

rvWrapSimple was used in TRichView versions prior to 13. It is faster than *rvWrapNormal*.

rvWrapNormal is the default and recommended value.

Example



Default value:

rvWrapNormal

1.4.2.1.1.22 TRVStyle.ListStyles

This is one of the key properties of TRVStyle: collection of paragraph list styles (bullets and numbering).

```
property ListStyles : TRVListInfos[425];
```

Bullets/numbering (lists) are implemented in the following way:

1. TRVStyle has a collection of list styles (this property). Each list style[459] contains a collection of list levels (Levels[459] property). A list level[472] defines the most of list properties.

2. List markers 99 is a special type of items. They are inserted with the special functions. They are always inserted at the beginning of paragraphs. They have the following properties:

- ListNo – index of list style in the collection of list styles (i.e. in this property);
- LevelNo – list level;
- StartFrom – starting value for list counter, if UseStartFrom=*True*;
- UseStartFrom – if *True*, list counter value for this marker is defined by StartFrom. If *False*, numbering is continued.

See also properties:

- TextStyles 404;
- ParaStyles 398.

See also types:

- TRVListInfos 425 (collection of list styles);
- TRVListInfo 459 (item in the collection of list styles).

See also methods of TCustomRichView:

- SetListMarkerInfo 233,
- GetListMarkerInfo 199.

See also methods of TCustomRichViewEdit:

- ApplyListStyle 297,
- RemoveLists 334,
- ChangeListLevels 304.

See also:

- List markers item type 99.

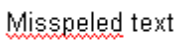
1.4.2.1.1.23 TRVStyle.LiveSpellingColor

Color of live spelling wavy underlines.

```
property LiveSpellingColor: TColor;
```

(introduced in v1.9)

Example:



Default value:

clRed

See also:

- Live spelling check in RichView 69.

1.4.2.1.1.24 TRVStyle.MainRVStyle

Defines the master RVStyle component for this component.

```
property MainRVStyle: TRVStyle;
```

(introduced in version 14)

If this **MainRVStyle** is assigned, StyleTemplates 402 returns **MainRVStyle**.StyleTemplates 402. It allows several editors working with the same collection of style templates.

If some RichView component is used as a header 276/footer 276 (and style templates are used 158) it must have Style 156.MainRVStyle assigned equal to Style 156 of the main editor.

ScaleRichView note: TSRichViewEdit provides that srve.RichViewEdit.Style = srve.RVHeader.Style.MainRVStyle = srve.RVFooter.Style.MainRVStyle.

1.4.2.1.1.25 TRVStyle.PageBreakColor

Color of explicit (user-defined) page breaks in TRichView components.

```
property PageBreakColor: TColor;
```

This color is used for drawing horizontal lines (with "folded paper" icon) specifying explicit (user-defined) page breaks. You can change this drawing in OnDrawPageBreak 413 event.

Explicit page breaks are added by assigning *True* to TRichView.PageBreaksBeforeItems 149[], or using TRichViewEdit.InsertPageBreak 321.

TRichView must be linked with this TRVStyle (see TRichView.Style 156).

Default value:

clBtnShadow

See also properties:

- SoftPageBreakColor 401.

See also events:

- OnDrawPageBreak 413.

See also properties of TCustomRichView:

- Options 146 (*rvoShowPageBreaks*).

1.4.2.1.1.26 TRVStyle.ParaStyles

This is one of the key properties of TRVStyle: collection of paragraph styles.

```
property ParaStyles : TParaInfos 423;
```

Paragraph styles define paragraph attributes in documents.

See also properties:

- TextStyles 404;
- ListStyles 396.

See also types:

- TParaInfos 423 (collection of paragraph styles);
- TParaInfo 448 (item in collection of paragraph styles).

See also:

- Paragraphs in RichView 37.

1.4.2.1.1.27 TRVStyle.SelectionHandleKind

Defines visual appearance of selection handles for touch screens

```
property SelectionHandleKind: TRVSelectionHandleKind;

type
  TRVSelectionHandleKind = (rvshkWedge, rvshkCircle);
```

(introduced in v15)

This property exists when the component is compiled in Delphi 2010 or newer.

Value	Appearance
rvshkWedge	

rvshkCircle	

rrvshkCircle is not recommended, because it may overlap selected text in table cells; you can use it in ScaleRichView, because TSRichViewEdit does not shift handles to fit in cells.

Default value:

rvshkWedge

See also properties of TCustomRichView:

- SelectionHandlesVisible 156

1.4.2.1.1.28 TRVStyle.SelectionMode

Defines how the selection is guided by TRichView components.

```
property SelectionMode: TRVSelectionMode;

type
  TRVSelectionMode =
   (rvsmChar, rvsmWord, rvsmParagraph);
```

(introduced in v 1.8)

Selection Mode	Meaning
rvsmChar	No special processing for selection: selection starts where the user pressed the left mouse button, selection ends where the user released it.
rvsmWord	Selecting by words. Selection is smart: you can still select a part of word by moving mouse pointer in backward direction. This mode affects only selecting by moving mouse.
rvsmParagraph	Special selecting mode: if items of multiple paragraphs are selected, the component selects these paragraphs completely. The mode affects selection by moving mouse, Shift+click, keyboard.

Default value:

rvsmWord

1.4.2.1.1.29 TRVStyle.SelectionStyle

Defines visual appearance of selection in TRichView components.

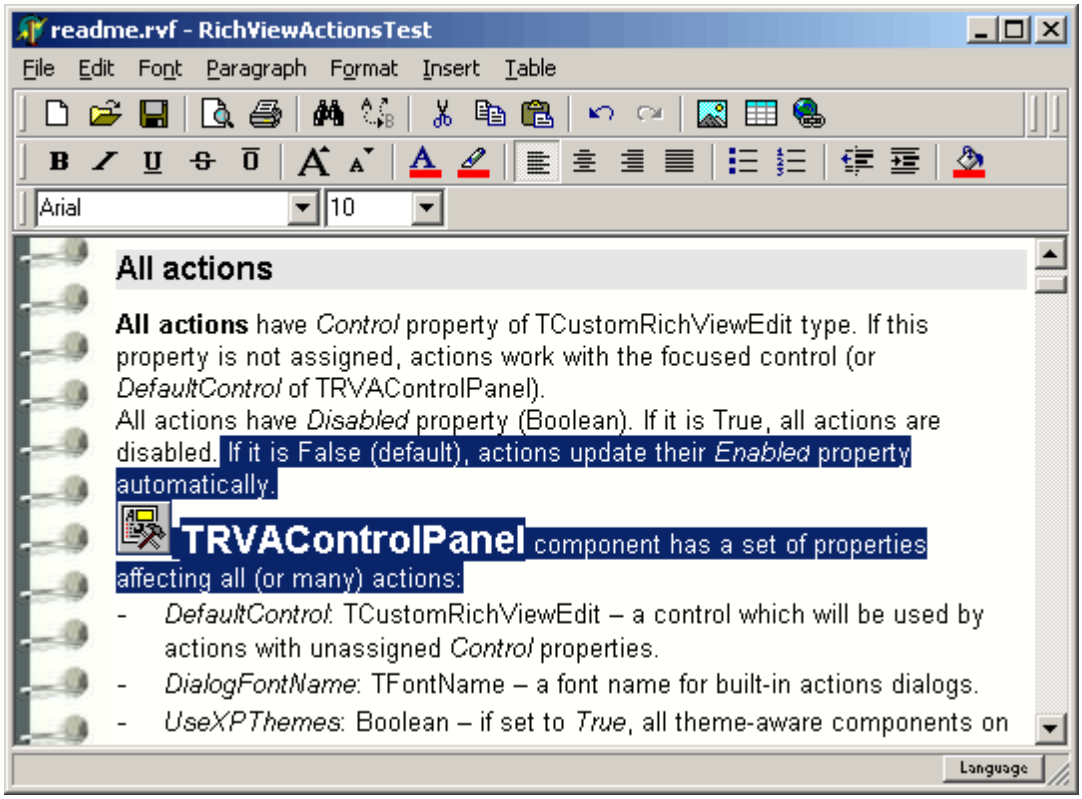
```
property SelectionStyle: TRVSelectionStyle;

type
  TRVSelectionStyle = (rvssItems, rvssLines);
```

(introduced in v 1.8)

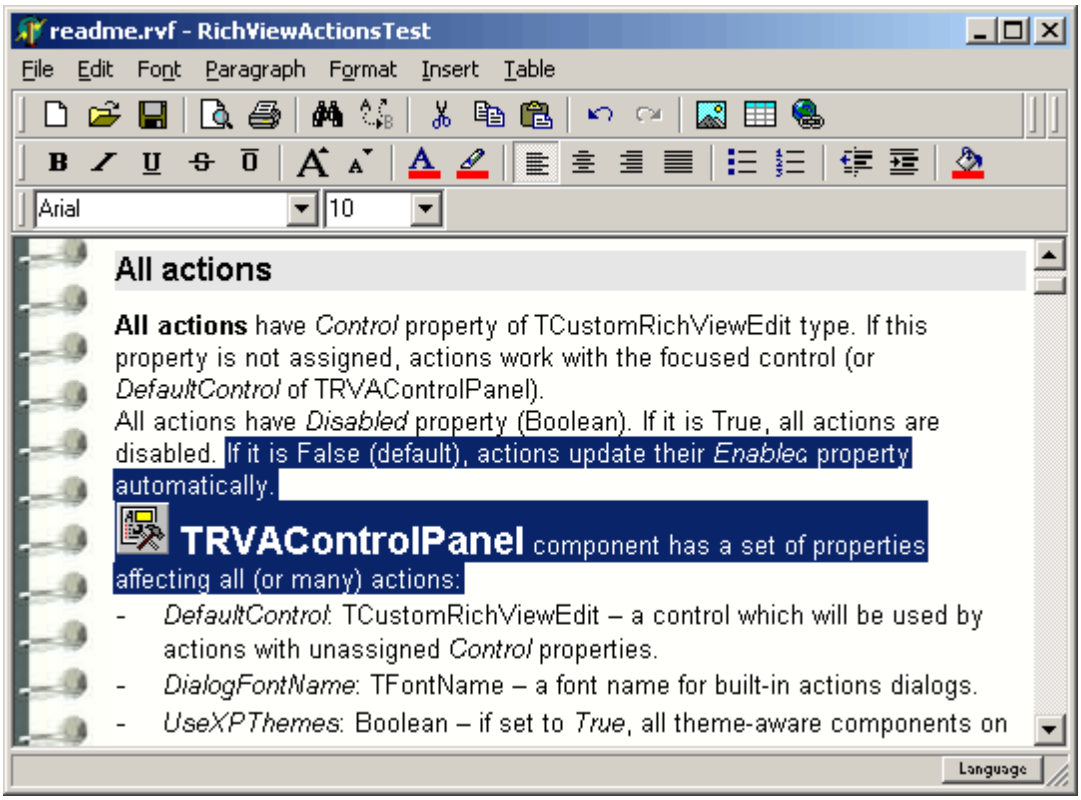
Selection Style	Meaning
rvssItems	Each item draws its selection separately.

rvssLines	The selection looks more like in MS Word (selection has height of the highest item in the line, including spacing); this mode works only if BiDiMode [69]=*rvbdUnspecified*.

rvssItems:



rvssLines:



Default value:

rvssItems

See also properties:

- SelColor, SelTextColor, InactiveSelColor, InactiveSelTextColor 393;
- SelectionMode 399.

1.4.2.1.1.30 TRVStyle.SoftPageBreakColor

Color of soft page breaks in TRichView components.

```
property PageBreakColor: TColor;
```

This color is used for drawing horizontal lines (with "folded paper" icon) specifying soft (automatic) page breaks. You can change this drawing in OnDrawPageBreak 413 event.

TRichView must be linked with this TRVStyle (see TRichView.Style 156). Soft page breaks displaying is turned on by TRichView.AssignPageBreaks 174.

Default value:

clBtnFace

See also properties:

- PageBreakColor 398.

See also events:

- OnDrawPageBreak 413.

See also properties of TCustomRichView:

- Options 146 (*rvoShowPageBreaks*).

1.4.2.1.1.31 TRVStyle.SpacesInTab

If set to positive value, TRichView components will replace TAB characters with the specified number of space characters.

```
property SpacesInTab: SmallInt;
```

This property is used in all methods of TRichView 128 and TRichViewEdit 282 for adding/inserting multiline text and RTF, or on typing.

TRichView must be linked with this TRVStyle (see TRichView.Style 156).

If this property is 0, TABs are inserted as a special tabulator item type 89.

Note: default value is changed to 0 since v1.9

Default value:

0

See also properties:

- DefTabWidth 388.

1.4.2.1.1.32 TRVStyle.SpecialCharactersColor

Color of special characters.

```
property SpecialCharactersColor: TColor;
```

(introduced in version 12)

if *rvoShowSpecialCharacters* is included in TRichView.Options 146, this color is used for drawing marks denoting space characters and paragraph breaks.

If it equals to *clNone*, a normal text color is used instead.

Default value:

clNone

See also:

- RVVisibleSpecialCharacters 665 global typed constant
- FloatingLineColor 390

1.4.2.1.1.33 TRVStyle.StyleTemplates

A collection of real styles, controlling text attributes from TextStyles 404 and paragraph attributes from ParaStyles 398.

```
property StyleTemplates : TRVStyleTemplateCollection 427;
```

Using style templates is optional, they are used only for richview controls having UseStyleTemplates 158=*True*.

If MainRVStyle 397 is assigned, this property returns MainRVStyle 397.**StyleTemplates**.

See also

- Styles and style templates 40.

1.4.2.1.1.34 TRVStyle.TextBackgroundKind

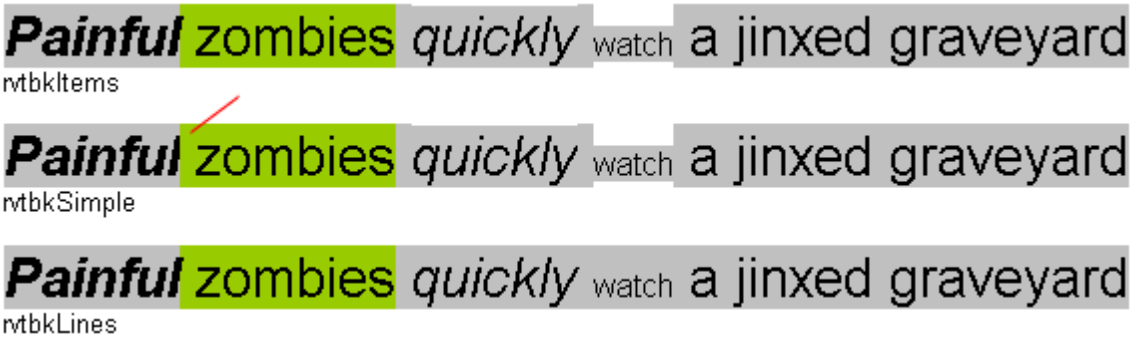
Defines visual appearance of text background in TRichView components.

```
type
  TRVTextBackgroundKind = (rvtbkSimple, rvtbkItems, rvtbkLines);
property TextBackgroundKind: TRVTextBackgroundKind;
```

(introduced in v14)

Selection Style	Background is drawn...	Background height =
rvtbkItems	before text	item height
rvtbkSimple	together with text	item height
rvtbkLines	before text	line height

Example:



rvtbkSimple is not recommended, because the item text may be overlapped by a background of the next item, especially for italic fonts (see the red line in the example above).

Default value:

rvtbkItems

See also:

- OnDrawTextBack 416

See also properties of text style:

- BackColor 433
- HoverBackColor 436;

1.4.2.1.1.35 TRVStyle.TextEngine

Specifies which set of functions are used for text measuring and drawing.

```
type
  TRVTextEngine = (rvteWindows, rvteUniscribe);
property TextEngine: TRVTextEngine;
```

(introduced in v16)

Value	Meaning	Comments
rvteWindows	Windows API	**Pros:** • faster, especially if the support of bidirected text is turned off. **Cons:** • completely incorrect results for right-to-left text (Arabic and Hebrew), if the support of bidirected text is turned off; • results may be not completely correct when text includes some "exotic" characters; • unreliable results when the support of bidirected text is turned on (may be correct for one font, but incorrect for another font).
rvteUniscribe	***Uniscribe***	**Pros:** • reliable processing of complex scripts and "exotic" characters;

		• acceptable results for right-to-left text even if support of bidirected text is turned off **Cons:** • usually slower

Reformat documents in linked TRichView controls after changing value of this property.

Default value:

rvteUniscribe

1.4.2.1.1.36 TRVStyle.TextStyles

This is one of key properties of TRVStyle: a collection of text attributes (styles).

```
property TextStyles : TFontInfos[418];
```

Text styles define fonts and other text attributes in documents.

The first text style in the collection (TextStyles[0]) has some additional meaning:

- *breaks*[29] with color = *clNone* (default value) have the color of this style;
- Charset[434] of this style is used for saving language information in HTML files (D3+);
- in some cases this style is used as default, so it's recommended to use it as a style of normal text; please do not set *rvprDoNotAutoSwitch* in Protection[439] for this style;
- this style can be hypertext, but it is not recommended.

See also properties:

- ParaStyles[398];
- ListStyles[396].

See also types:

- TFontInfos[418] (collection of text styles);
- TFontInfo[444] (item in collection of text styles).

1.4.2.1.1.37 TRVStyle.Units

Defines units of measurement for properties of TRVStyle and linked[156] controls.

```
property Units: TRVStyleUnits[644]
```

(introduced in version 13)

All values of type TRVStyleLength[644] are measured in units specified in this property.

Assignment to this property does not convert properties to new units. To change units of measurement, at first call one of ConvertDocTo*[176] methods of all TRichView[128] components linked[156] to this TRVStyle, then call one of ConvertTo*[405] methods to convert properties of this TRVStyle component to the same units of measurement.

At design time, you can convert units using TRVStyle's context menu ("Convert to Pixels" or "Convert to Twips" commands).

Default value:

rvstuPixels

See also properties:

- Units of measurement in TRichView[74] (contains a list of properties measured in Units)

1.4.2.1.1.38 TRVStyle.UseSound

Defines whether RichViewEdit[282] will use sound.

```
property UseSound: Boolean
```

(introduced in version 1.3)

If *True*, RichViewEdit plays a system default sound when user tries to modify read-only [294] or protected [439] text.

Default value:

True

1.4.2.1.2 Methods

1.4.2.1.2.1 TRVStyle.AddTextStyle

```
function AddTextStyle: Integer;
```

Obsolete. In order to add item in collection of text styles, use RVStyle.TextStyles.Add [419]

1.4.2.1.2.2 TRVStyle.Create

A constructor, creates a new TRVStyle component.

```
constructor Create(AOwner: TComponent);override;
```

Use Create to instantiate TRVStyle component at runtime. TRVStyle components placed on forms at design time are created automatically. Specify the owner of the new RVStyle using the AOwner parameter.

1.4.2.1.2.3 TRVStyle.ConvertTo*

The methods convert all properties of this TRVStyle component to new units of measurement.

```
procedure ConvertToPixels;
procedure ConvertToTwips;
procedure ConvertToDifferentUnits(NewUnits: TRVStyleUnits [644]);
```

(introduced in version 13)

The methods convert all properties of this TRVStyle (including all sub-properties) from Units [404] to:

- screen pixels [74],
- twips [74],
- NewUnits (twips or pixels),

respectively.

After the conversion, a new value is assigned to Units [404] property.

If there is a document in TRichView control linked to this TRVStyle component, this document must be converted before calling this method.

When converting to/from pixels, RichViewPixelsPerInch [655] is taken into account.

1.4.2.1.2.4 TRVStyle.DeleteTextStyle

```
procedure DeleteTextStyle(Index: Integer);
```

Obsolete. In order to delete the i-th item from text styles, use TRVStyle.TextStyles [404][i].Free

1.4.2.1.2.5 TRVStyle.Destroy

A destructor, frees an instance of a TRVStyle component,

```
destructor Destroy;override;
```

Destroy seldom needs to be called. When TRVStyle is constructed without an owner by the Create method, Free should be called to release memory and dispose the object. Free checks to see if the pointer is nil before calling Destroy.

1.4.2.1.2.6 TRVStyle.GetAs*

The methods return **Value** converted to different units of measurement.

```
function GetAsPixels(Value: TRVStyleLength644): Integer;
function GetAsTwips(Value: TRVStyleLength644): Integer;
function GetAsDifferentUnits(Value: TRVStyleLength644; Units: TRVStyleUnits644): TRVStyleLen
function GetAsRVUnits(Value: TRVStyleLength644; RVUnits: TRVUnits648): TRVLength636;
```

(introduced in version 13)

The methods return **Value** converted from Units404 to

- screen pixels74,
- twips74,
- another Units (twips or pixels),
- RVUnits (inches, mm, points, etc.)

respectively.

When converting to/from pixels, RichViewPixelsPerInch655 is taken into account.

See also:

- *ToUnits406 methods;
- ConvertTo*405 methods.

1.4.2.1.2.7 TRVStyle.*ToUnits

The methods return **Value** converted from different units of measurement.

```
function PixelsToUnits(Value: Integer): TRVStyleLength644;
function TwipsToUnits(Value: Integer): TRVStyleLength644;
function DifferentUnitsToUnits(Value: TRVStyleLength644; Units: TRVStyleUnits644): TRVStyleL
function RVUnitsToUnits(Value: TRVLength636; RVUnits: TRVUnits648): TRVStyleLength644;
```

(introduced in version 13)

The methods return **Value** converted to Units404 from

- screen pixels74,
- twips74,
- another Units (twips or pixels),
- RVUnits (inches, mm, points, etc.)

respectively.

When converting to/from pixels, RichViewPixelsPerInch655 is taken into account.

See also:

- GetAs*406 methods;
- ConvertTo*405 methods.

1.4.2.1.2.8 TRVStyle.GetListStyleClass

Returns the class type representing a style of paragraph lists (bullets and numbering).

```
type
  TRVListInfoClass = class of TRVListInfo459;

function GetListStyleClass: TRVListInfoClass; virtual;
```

(introduced in version 1.7)

This method can be used to create components inherited from TRVStyle. You can create a new class (inherited from TRVListInfo459) and use it in this descendant component.

See also:

- GetParaStyleClass 407;
- GetTextStyleClass 407.

1.4.2.1.2.9 TRVStyle.GetParaStyleClass

Returns the class type representing a style of paragraphs.

```
type
  TRVParaInfoClass = class of TParaInfo 448;

function GetParaStyleClass: TRVParaInfoClass; virtual;
```

(introduced in version 1.7)

This method can be used to create components inherited from TRVStyle. You can create a new class (inherited from TParaInfo 448) and use it in this descendant component.

See also:

- GetListStyleClass 406;
- GetTextStyleClass 407.

1.4.2.1.2.10 TRVStyle.GetTextStyleClass

Returns the class type representing a style of text.

```
type
  TRVFontInfoClass = class of TFontInfo 444;

function GetTextStyleClass: TRVFontInfoClass; virtual;
```

(introduced in version 1.7)

This method can be used to create components inherited from TRVStyle. You can create a new class (inherited from TFontInfo 444) and use it in this descendant component.

See also:

- GetListStyleClass 406;
- GetParaStyleClass 407.

1.4.2.1.2.11 TRVStyle.LoadINI

Loads all properties of TRVStyle component from ini-file

```
procedure LoadINI(const FileName, Section: String);
```

Parameters:

FileName – name of the ini-file;

Section – section of the ini-file.

See also:

- SaveINI 409;
- LoadReg 408.

1.4.2.1.2.12 TRVStyle.FindTextStyle, FindParaStyle

The methods search for a style like the style specified in the parameter, and add it if necessary.

```
function FindTextStyle(TextStyle: TFontInfo 444): Integer;
function FindParaStyle(ParaStyle: TParaInfo 456): Integer;
```

(introduced in version 14)

The methods are equivalent to:

```
TextStyles[404].FindSuchStyleEx[422](0, TextStyle, RVAllFontInfoProperties[633])
ParaStyles[398].FindSuchStyleEx[425](0, ParaStyle, RVAllParaInfoProperties[637])
```

1.4.2.1.2.13 TRVStyle.LoadReg

Loads all properties of TRVStyle component from the Registry (for Delphi4+).

```
procedure LoadReg(const BaseKey: String);
```

(Introduced in version 1.2)

Parameter:

BaseKey – the name of the HKEY_CURRENT_USER subkey to loads data from. LoadReg appends 'RVStyle' to this subkey. For example, if **BaseKey** is 'Software\My Company\My Program', it loads data from the 'Software \My Company\My Program\RVStyle' key.

Warning: these data in this section (BaseKey+'\RVStyle') must exists, otherwise all properties will be reset to default values.

See also:

- SaveReg[409];
- LoadINI[407].

1.4.2.1.2.14 TRVStyle.SaveCSS

Saves text and paragraph styles as CSS (Cascading Style Sheets) file.

```
type
  TRVSaveCSSOption = (
    rvcssOnlyDifference, rvcssIgnoreLeftAlignment,
    rvcssNoDefCSSStyle, rvcssUTF8, rvcssDefault0Style,
    rvcssMarkersAsText);
  TRVSaveCSSOptions = set of TRVSaveCSSOption;

function SaveCSS(const FileName: String;
  Options: TRVSaveCSSOptions): Boolean;
```

Use this method if you want to save content of TRichView as HTML file (RichView.SaveHTMLEx[217]) with external CSS.

You can save CSS only once and use it for several HTML files (if they are linked to the same or compatible RVStyle components) .

This method allows to reduce size of HTML files.

The following options may be included in the **Options** parameter.

Option	Meaning
rvcssNoDefCSSStyle	Corresponds to *rvsoNoDefCSSStyle*[639] in the Options parameters of SaveHTMLEx[217]. It's not recommended to use this option.
rvcssDefault0Style	Properties of TextStyles[404][0] are not saved. Corresponds to *rvsoDefault0Style*[639] in the Options parameter of SaveHTMLEx[217].
rvcssMarkersAsText	Lists will saved as text, so this method creates an additional styles for text bullets. Corresponds to *rvsoMarkersAsText*[639] in the Options parameter of SaveHTMLEx[217].

rvcssUTF8	CSS is saved in UTF-8 encoding.

Please do not use other options.

There are no methods for loading CSS files.

See also:

- RichView work with HTML files [62].

1.4.2.1.2.15 TRVStyle.SaveCSSToStream

The same as SaveCSS [408], but accepts **Stream** parameter

```
procedure SaveCSSToStream(Stream: TStream; Options: TRVSaveCSSOptions);
```

(introduced in v1.1.4)

1.4.2.1.2.16 TRVStyle.SaveINI

Saves all properties of TRVStyle component in the ini-file

```
procedure SaveINI(const FileName, Section: String);
```

Parameters:

FileName – name of the ini-file;

Section – section of the ini-file

Warning: the section is cleared before saving.

Alternative way to store/load TRVStyle: using TStream.WriteComponent and ReadComponent.

See also:

- LoadINI [407];
- SaveReg [409];
- SaveCSS [408].

1.4.2.1.2.17 TRVStyle.SaveReg

Saves all properties of component to the Registry (for Delphi4+)

```
procedure SaveReg(const BaseKey: String);
```

(Introduced in version 1.2)

Parameter:

BaseKey – the name of the HKEY_CURRENT_USER subkey to saves data to. SaveReg appends 'RVStyle' to this subkey. For example, if **BaseKey** is 'Software\My Company\My Program', the method saves data to the 'Software\My Company\My Program\RVStyle' key.

Warning: all data in this key (BaseKey+'\RVStyle') will be removed before saving!

The saving format is the same as in SaveINI [409].

See also:

- LoadReg [408];
- SaveINI [409].

1.4.2.1.3 Events

1.4.2.1.3.1 TRVStyle.OnAfterApplyStyle

Occurs after RichView applies attributes of the **StyleNo**-th style to Canvas.

```
property OnAfterApplyStyle: TRVAfterApplyStyleEvent;

type
  TRVApplyStyleEvent =
    procedure (Sender: TRVStyle; Canvas: TCanvas;
      StyleNo: Integer) of object;
```

(introduced in version 12)

In this event you can apply font (excluding colors) for the text style **StyleNo** to **Canvas**. Nothing should be drawn inside this event.

This event is similar to OnApplyStyle [410], but occurs after the default procedure. This event does not occur if OnApplyStyle [410] returned *False* in DoDefault parameter.

Input parameters:

Sender – TRVStyle generating this event;

Canvas – canvas for applying font attributes to.

See also:

- OnApplyStyleColor [411];
- OnDrawStyleText [415].

See also properties:

- TextStyles [404].

1.4.2.1.3.2 TRVStyle.OnApplyStyle

Occurs when RichView applies attributes of the **StyleNo**-th style to Canvas.

```
property OnApplyStyle: TRVApplyStyleEvent;

type
  TRVApplyStyleEvent =
    procedure (Sender: TRVStyle; Canvas: TCanvas;
      StyleNo: Integer; var DoDefault: Boolean) of object;
```

(introduced in version 1.3)

In this event you can apply font (excluding colors) for the text style **StyleNo** to **Canvas**.

Nothing should be drawn inside this event.

Input parameters:

Sender – TRVStyle generating this event;

Canvas – canvas for applying font attributes to.

DoDefault is equal to True.

Output parameters:

DoDefault – set to *False* to prevent the default style applying procedure and OnAfterApplyStyle [410] event.

See also:

- OnApplyStyleColor [411];
- OnDrawStyleText [415].

See also properties:

- TextStyles 404.

1.4.2.1.3.3 TRVStyle.OnApplyStyleColor

Occurs when RichView applies the color of the **StyleNo**-th style to **Canvas**.

```
property OnApplyStyleColor: TRVApplyStyleColorEvent;

type
  TRVApplyStyleColorEvent =
    procedure (Sender: TRVStyle; Canvas: TCanvas;
      StyleNo: Integer; DrawState: TRVTextDrawStates 645;
      var DoDefault: Boolean) of object;
```

(introduced in version 1.3)

In this event you can apply font color for the **StyleNo**-th text style to **Canvas**.

Nothing should be drawn inside this event.

Input parameters:

Sender – TRVStyle generating this event.

Canvas – canvas for applying background and text colors to.

DrawState – states of this text item, see TRVTextDrawStates 645.

DoDefault is equal to *True*.

Output parameters:

DoDefault – set to *False* to prevent the default colors applying procedure.

See also:

- OnApplyStyle 410;
- OnDrawStyleText 415.

See also properties:

- TextStyles 404.

1.4.2.1.3.4 TRVStyle.OnDrawCheckpoint

Occurs when drawing *checkpoint* 30.

```
property OnDrawCheckpoint: TRVDrawCheckpointEvent;

type
  TRVDrawCheckpointEvent =
    procedure (Sender: TRVStyle; Canvas: TCanvas;
      X,Y, ItemNo, XShift: Integer;
      RaiseEvent: Boolean; Control: TControl;
      var DoDefault: Boolean) of object;
```

(introduced in version 1.3)

You can use this event to draw your own visual representation of *checkpoints*.

This event occurs only if *rvoShowCheckpoints* is included in TCustomRichView(**Control**).Options 146.

Input parameters:

Sender – TRVStyle generating the event.

Canvas – canvas to paint.

X,**Y** – coordinates of the top left corner of the item marked with this *checkpoint*. If this is the last *checkpoint* in document (that is not associated with any item), **X** = -1.

ItemNo – index of this item in **Sender**.**RVData**. (**ItemNo** = -1 for *checkpoint* without item).

XShift – how much TRichView is scrolled horizontally (>=0);

RaiseEvent – "raise event" flag of the *checkpoint*.

Control – TRichView control.

DoDefault is equal to *True*.

"Hidden" input parameter:

Sender.**RVData**: TPersistent, document containing the item with this *checkpoint*. It should be typecasted to TCustomRVFormattedData[608] before using.

Output parameters:

DoDefault – set to *False* to prevent default drawing. The default procedure draws horizontal dotted line with small circle in (**X**,**Y**). The line color is either CheckpointColor or CheckpointEvColor[387], depending on **RaiseEvent**.

Using **ItemNo** and **Sender**.**RVData** you can get additional information about the item and the *checkpoint*.

For example:

```
uses ..., CRVFData;
...
var
  CheckpointData: TCheckpointData[622];
  CheckpointTag: TRVTag[645];
  CheckpointName: String;
begin
  if ItemNo>=0 then
  begin
    CheckpointData :=
      TCustomRVFormattedData[608](Sender.RVData).GetItemCheckpoint[191](ItemNo));
    TCustomRVFormattedData(Sender.RVData).GetCheckpointInfo[185](
      CheckpointTag, CheckpointName, RaiseEvent);
    ...
  end;
end;
```

See also properties:

- CheckpointColor, CheckpointEvColor[387].

See also:

- Checkpoints overview[30].

Demo project:

- Demos\Delphi\Assorted\Custom Draw\CustomDraw\
- Demos\CBuilder\Assorted\Custom Draw\CustomDraw\

1.4.2.1.3.5 TRVStyle.OnDrawPageBreak

Occurs when drawing page breaks.

```
property OnDrawPageBreak: TRVDrawPageBreakEvent;

type
  TRVPageBreakType = (rvpbSoftPageBreak, rvpbPageBreak);

  TRVDrawPageBreakEvent =
    procedure (Sender: TRVStyle;
      Canvas: TCanvas;
      Y, XShift: Integer;
      PageBreakType: TRVPageBreakType;
      Control: TControl;
      var DoDefault: Boolean) of object;
```

(introduced in version 1.3; modified in 1.7)

You can use this event to draw your own visual representation of page breaks.

This event occurs only if *rvoShowPageBreaks* is included in TCustomRichView(**Control**).Options [146].

Input parameters:

Sender – TRVStyle generating the event.

Canvas – canvas to paint.

PageBreakType – type of page break (either soft (automatic) page break or explicit (user-defined) page break).

Y – y coordinate of page break in TRichView window.

XShift – how much TRichView is scrolled horizontally (>=0).

Control – TRichView control.

DoDefault is equal to *True*.

"Hidden" input parameters:

Sender.**RVData**: TPersistent, document where page break occurs. It should be typecasted to TCustomRVFormattedData [608] before using.

Sender.**ItemNo**: Integer, index of the item in **Sender**.**RVData**.

These parameters define one of the following:

- item starting a new page;
- table [528] with page breaks between rows.

Output parameter:

DoDefault – set to *False* to prevent default drawing. The default procedure draws horizontal line horizontal line with "dog-ear" effect. The line color is either PageBreakColor [398] or SoftPageBreakColor [401], depending on **PageBreakType**.

Using **Sender**.**RVData** you can get additional information about the item:

For example:

```
uses ..., CRVFData;
...
var ParaNo: Integer;
begin
  ParaNo := TCustomRVFormattedData[608](Sender.RVData).GetItemPara[194](Sender.ItemNo));
```

```
  ...
end;
```

See also properties:

- PageBreakColor 398;
- SoftPageBreakColor 401.

See also properties of TCustomRichView 128:

- PageBreaksBeforeItem 149.

See also methods of TCustomRichView 128:

- AssignSoftPageBreaks 174.

See also methods of TCustomRichViewEdit 282:

- InsertPageBreak 321;
- RemoveCurrentPageBreak 334.

See also properties of table 528:

- PrintOptions 542.

See also properties of table row 577:

- PageBreakBefore 578.

Demo project:

- Demos\Delphi\Assorted\Custom Draw\CustomDraw\
- Demos\CBuilder\Assorted\Custom Draw\CustomDraw\

1.4.2.1.3.6 TRVStyle.OnDraw ParaBack

Occurs when painting a background of paragraph of the **ParaNo**-th style.

```
property OnDrawParaBack: TRVDrawParaRectEvent;

type
  TRVDrawParaRectEvent =
    procedure (Sender: TRVStyle; Canvas: TCanvas;
      ParaNo: Integer; ARect: TRect;
      var DoDefault: Boolean) of object;
```

(introduced in version 1.3)

You can use this event to draw your own background and/or border for some or all paragraphs.

Input parameters:

Sender – TRVStyle generating the event.

Canvas – canvas to paint.

ARect – rectangle where to paint.

ParaNo – index in the collection of paragraph styles (ParaStyles 398).

DoDefault is equal to *True*.

"Hidden" input parameters:

Sender.**RVData**: TPersistent, document containing this paragraph. It should be typecasted to TCustomRVFormattedData 608 before using.

Sender.**ItemNo**: Integer, index of the last item in the paragraph, in **Sender**.**RVData**.

Output parameters:

DoDefault – set to *False* to prevent default drawing.

See also properties:

- ParaStyles 398;
- TParaInfo 448.Background 450.

See also:

- RichView Paragraphs 37.

Demo project:

- Demos\Delphi\Assorted\Custom Draw\CustomDraw\
- Demos\CBuilder\Assorted\Custom Draw\CustomDraw\

1.4.2.1.3.7 TRVStyle.OnDrawStyleText

Allows to perform custom drawing of a text item of the **StyleNo**-th text style.

```
property OnDrawStyleText: TRVDrawStyleTextEvent

type
  TRVDrawStyleTextEvent = procedure (Sender: TRVStyle 385;
    const s: TRVRawByteString 638; Canvas: TCanvas; StyleNo: Integer;
    SpaceAtLeft, Left, Top, Width, Height
    {$IFDEF RVUSEBASELINE},BaseLine{$ELSE}{$IFDEF RVHIGHFONTS},DrawOffsY{$ENDIF}{$ENDIF}:
    TextWidth, SpaceCount: Integer;
    DrawState: TRVTextDrawStates 645;
    Printing, PreviewCorrection, ForMetafile: Boolean;
    ColorMode: TRVColorMode 625; DefBiDiMode: TRVBiDiMode 623;
    RefCanvas: TCanvas;
    LeftNoExpIndex, RightNoExpIndex: Integer;
    var DoDefault: Boolean) of object;
```

(introduced in version 1.3; new states in TRVTextDrawStates 645 since 1.4; new parameters since v16)

You can use this event to modify drawing of text of some or all text styles.

Input parameters:

The same is in TFontInfo 444.Draw 447. You can call **Sender**.TextStyles 404[**StyleNo**].Draw(..) passing the even parameters.

"Hidden" input parameters:

Sender.**RVData**: TPersistent, document containing this paragraph. It should be typecasted to TCustomRVFormattedData 608 before using.

Sender.**ItemNo**: Integer, index of the item in **Sender**.**RVData**.

Self.**OffsetInItem**: Integer, index of the first character in the item to draw.

These parameters allow to get additional information about the item being painted.

For example:

```
uses ..., CRVFData;
...
var ItemTag: TRVTag 645;
...
ItemTag := TCustomRVFormattedData 608(Sender.RVData).GetItemTag 195(Sender.ItemNo));
```

Output parameters:

DoDefault – set to *False* to prevent default drawing.

See also events:

- OnApplyStyle[410];
- OnApplyStyleColor[411];
- OnDrawTextBack[416].

See also properties:

- TextStyles[404].

Demo projects:

- Demos*\Assorted\Custom Draw\

1.4.2.1.3.8 TRVStyle.OnDrawTextBack

Occurs when drawing background of text of the **StyleNo**-th text style.

```
property OnDrawTextBack: TRVDrawTextBackEvent;

type
  TRVDrawTextBackEvent =
    procedure (Sender: TRVStyle; Canvas: TCanvas;
      StyleNo: Integer; Left, Top, Width, Height: Integer;
      DrawState: TRVTextDrawStates[645];
      var DoDefault: Boolean) of object;
```

(introduced in version 1.3)

You can use this event to modify drawing of text background for some or all text styles.

Unlike OnDrawStyleText[415], this event is called only once for each drawing item (OnDrawStyleText[415] can be called several times if part of text item is selected). In paragraphs having Alignment[449]=*rvaJustify*, each word is a drawing item, but OnDrawTextBack is called only once per line, covering all area below this text item.

Input parameters:

Sender – TRVStyle generating the event.

Canvas – canvas to paint.

StyleNo – index of text style in TextStyles[404] collection.

Left, **Top**, **Width**, **Height** – rectangle where to paint.

DrawState may include *rvtsControlFocused* and *rvtsHover*, see , see TRVTextDrawStates[645].

DoDefault is equal to *True*.

"Hidden" input parameters:

Sender.**RVData**: TPersistent, document containing this paragraph. It should be typecasted to TCustomRVFormattedData[608] before using.

Sender.**ItemNo**: Integer, index of the item in **Sender**.**RVData**.

These parameters allow to get additional information about the item being painted.

For example:

```
uses ..., CRVFData;
...
var ItemTag: TRVTag[645];
...
ItemTag := TCustomRVFormattedData[608](Sender.RVData).GetItemTag[195](Sender.ItemNo));
```

Output parameters:

DoDefault – set to *False* to prevent default drawing. It is supported only if TextBackgroundKind[402] <>*rvtbkSimple*.

See also events:

- OnDrawStyleText[415]
- OnApplyStyle[410];
- OnApplyStyleColor[411];

See also properties:

- TextStyles[404].

Demo projects:

- Demos\Delphi\Assorted\Custom Draw\
- Demos\CBuilder\Assorted\Custom Draw\

1.4.2.1.3.9 TRVStyle.OnStyleHoverSensitive

In this event you can define whether TRichView should redraw itself when the mouse pointer moves in or out of the area of text of the **StyleNo**-th style.

```
property OnStyleHoverSensitive: TRVStyleHoverSensitiveEvent;

type
  TRVStyleHoverSensitiveEvent =
    procedure (Sender: TRVStyle;
      StyleNo: Integer;
      var Sensitive: Boolean) of object;
```

(introduced in version 1.3)

This event occurs only for hypertext[445] styles.

Input parameters:

Sender – TRVStyle generating the event.

StyleNo – index of text style in TextStyles[404] collection.

Sensitive – *True* if for the given text style (HoverEffects[437]=[]) or (HoverBackColor[436]<>BackColor[433]) or (HoverColor[436]<>*clNone*) or (HoverUnderlineColor[437]<>*clNone*), or if HoverColor[393]<>*clNone*. *False* otherwise.

Output parameters:

Set **Sensitive** to *True* is you want to draw a special effect for "hot" (under the mouse pointer) text for this text style.

See also events:

- OnDrawStyleText[415];
- OnDrawTextBack[416].

See also properties:

- TextStyles[404].

1.4.2.1.4 Classes of Properties - Style Collections

1.4.2.1.4.1 TFontInfos

Collection of text attributes (styles). Class of TRVStyle.TextStyles[404] property.

Unit RVStyle;

Syntax

```
TCustomRVInfos = class(TCollection)
TFontInfos = class(TCustomRVInfos)
```

Hierarchy

TObject
TPersistent
TCollection
TCustomRVInfos

Main Properties

Items[418] – array of TFontInfo[444].

InvalidItem[418] is returned when accessing item which is out of index range.

Methods for Text Style Search

- FindStyleWithFontStyle[421]
- FindStyleWithFontSize[421]
- FindStyleWithColor[420]
- FindStyleWithFontName[421]
- FindSuchStyle, FindSuchStyleEx[422]
- FindStyleWithFont[420]
- FindStyleWithCharset[420]

See Also

- TParaInfos[423] – class of collection of paragraph attributes (styles);
- TRVListInfos[425] – class of collection of list attributes (i.e. styles of bullets and numbering).

This item is returned when accessing Items[418][ItemNo], if (ItemNo<0) or (ItemNo>=TFontInfos.Count)

```
property InvalidItem: TFontInfo[444];
```

(introduced in version 1.7)

This feature is for debugging, this situation should not occur.

Default value:

Default value has all properties Items[418][0], but red background[433] and white text color[435].

See also:

- TParaInfos.InvalidItem[423].

Provides indexed access to the items in the collection of text attributes (styles)

```
property Items[Index:Integer]: TFontInfo[444];
```

This is the default property of TFontInfos, so you can write

```
MyRVStyle.TextStyles[404][i]
```

instead of

```
MyRVStyle.TextStyles.Items[i]
```

Many methods of components inherited from TCustomRichView[128] have StyleNo parameter, which is an index

in TFontInfos.Items collection (more specifically, in TCustomRichView.Style[156].TextStyles[404].Items).

Example:

```
  MyRichView.AddNL[167]('Hello', 0, 0);
```

Unlike normal collections, when accessing item with invalid Index (for example, Index<0 or Index>=Count), no exception occurs. InvalidItem[418] is returned instead.

Indicates the number of pixels that make up a logical inch in the vertical direction.

```
PixelsPerInch: Integer;
```

This property affects how text is displayed in components inherited from TCustomRichView[128] and TRVReportHelper[503] (this property is ignored by TRVPrint[479]).

If this property is equal to 0, Screen.PixelsPerInch is used to display text.

You can assign positive value to this property to make your application independent of screen font mode. For example, if you assign 96, text will always look like in "Small Fonts" mode; if you assign 120, text will always look like in "Large Fonts" mode.

This property does not affect non-text elements, such as line widths or picture sizes (see RichViewPixelsPerInch[655] typed constant).

Default value:

0

Adds a new style in the collection of text styles

```
function Add: TFontInfo[444];
```

This method overrides TCollection.Add method to return TFontInfo instance.

Call Add to create a text style in the collection. The new item is placed at the end of the Items[418] array.

Return value:

Created collection item.

See also:

- TFontInfo.Create[447].

Adds a new item in the collection of text styles, initializes it.

```
function AddFont(Name: TFontName; Size: Integer;
  Color, BackColor: TColor; Style:TFontStyles): TFontInfo[444];
```

Please use Add[419] instead of this method.

Adds a new item in the collection of text styles, initializes it.

```
function AddFontEx(Name: TFontName; Size: Integer;
  Color, BackColor: TColor; Style:TFontStyles;
  Charset: TFontCharset): TFontInfo[444];
```

Please use Add[419] instead of this method.

Copies properties of this collection to the destination object **Dest**.

```
procedure AssignTo(Dest: TPersistent); override;
```

TCollection AssignTo method is overridden to make it possible to assign names of styles (TFontInfo.StyleName[432]) to TStrings.

Do not call AssignTo directly, this method is called from **Dest**.Assign method.

Example:

```
MyListBox: TListBox;
...
MyListBox.Items.Assign(MyRVStyle.TextStyles[404]);
```

A constructor, creates and initializes a collection of text styles.

```
constructor Create(Owner: TPersistent);
```

Do not create collection of styles yourself. It is created and destroyed by TRVStyle component as TextStyles[404] property.

Searches for style with the specified **Charset**.

```
function FindStyleWithCharset(BaseStyle: Integer; Charset: TFontCharset): Integer;
```

(Introduced in version 1.4)

Searches for style having all properties of the **BaseStyle**-th style, but Unicode[446]=*False* and Charset[434] =**Charset**.

BaseStyle is an index in Items[418].

The following properties are ignored when comparing styles:

- Standard[431];
- StyleName[432] (if RichViewCompareStyleNames[653]=*False* (default)).

Return value:

Index of style, or -1 if not found.

See also:

- other TFontInfos.**Find***** methods; the most universal method is FindSuchStyle[422];
- TFontInfo.IsEqual[444].

Searches for style with the specified colors.

```
function FindStyleWithColor(BaseStyle: Integer; Color, BackColor: TColor): Integer;
```

(Introduced in version 1.3)

Searches for style having all properties of the **BaseStyle**-th style, but Color[435]=**Color** and BackColor[433] =**BackColor**.

BaseStyle is an index in Items[418].

The following properties are ignored when comparing styles:

- Standard[431];
- StyleName[432] (if RichViewCompareStyleNames[653]=*False* (default)).

Return value:

Index of style, or -1 if not found.

See also:

- other TFontInfos.**Find***** methods; the most universal method is FindSuchStyle[422];
- TFontInfo.IsEqual[444].

Searches for style having the specified font attributes.

```
function FindStyleWithFont(BaseStyle: Integer; Font: TFont): Integer;
```

(Introduced in version 1.3)

Searches for style having all properties of the **BaseStyle**-th style, except for all font attributes defined in **Font**.

BaseStyle is an index in Items [418].

The following properties are ignored when comparing styles:

- Standard [431];
- StyleName [432] (if RichViewCompareStyleNames [653]=*False* (default)).

Return value:

Index of style, or -1 if not found.

See also:

- other TFontInfos.**Find***** methods; the most universal method is FindSuchStyle [422];
- TFontInfo.IsEqual [444].

Searches for style having specified font name.

```
function FindStyleWithFontName(BaseStyle: Integer; const FontName: TFontName): Integer;
```

(Introduced in version 1.3)

Searches for style having all properties of the **BaseStyle**-th style, but FontName [436]=**FontName**.

BaseStyle is an index in Items [418].

The following properties are ignored when comparing styles:

- Standard [431];
- StyleName [432] (if RichViewCompareStyleNames [653]=*False* (default)).

Return value:

Index of style, or -1 if not found.

See also:

- other TFontInfos.**Find***** methods; the most universal method is FindSuchStyle [422];
- TFontInfo.IsEqual [444].

Searches for style having the specified font size.

```
function FindStyleWithFontSize(BaseStyle: Integer; Size: Integer): Integer;
```

(Introduced in version 1.3)

Searches for style having all properties of the **BaseStyle**-th style, but font Size [441]=**Size**.

This method does not return items having odd values of SizeDouble [441].

BaseStyle is an index in Items [418].

The following properties are ignored when comparing styles:

- Standard [431];
- StyleName [432] (if RichViewCompareStyleNames [653]=*False* (default)).

Return value:

Index of style, or -1 if not found.

See also:

- other TFontInfos.**Find***** methods; the most universal method is FindSuchStyle [422];
- TFontInfo.IsEqual [444].

Searches for style having the specified font style.

```
function FindStyleWithFontStyle(BaseStyle: Integer; Value, Mask: TFontStyles): Integer;
```

(Introduced in version 1.3)

Searches for style having all properties of the **BaseStyle**-th style, but the font styles defined by **Value** and

Mask.

Mask is a set of styles for check.

Value is a set of styles what must exist.

BaseStyle is an index in Items[418].

For example:

- **Mask** = [*fsBold*], **Value** = [*fsBold*] searches for bold style (all other font styles are not checked).
- **Mask** = [*fsBold*], **Value** = [] searches for not bold style (all other font styles are not checked).

The following properties are ignored when comparing styles:

- Standard[431];
- StyleName[432] (if RichViewCompareStyleNames[653]=*False* (default)).

Return value:

Index of style, or -1 if not found.

See also:

- other TFontInfos.**Find***** methods; the most universal method is FindSuchStyle[422];
- TFontInfo.IsEqual[444].

Searches for a style.

```
function FindSuchStyle(BaseStyle: Integer; Style: TFontInfo[444];
  Mask: TRVFontInfoProperties[633]): Integer;
function FindSuchStyleEx(BaseStyle: Integer; Style: TFontInfo[444];
  Mask: TRVFontInfoProperties[633]): Integer;
```

(Introduced in version 1.3 and version 14)

Searches for a style having properties like **Style**.

The **BaseStyle**-th style is checked first. **BaseStyle** is an index in Items[418].

Mask is a set of properties to compare (other properties are ignored). Use RVAllFontInfoProperties[633] constant to compare all properties.

The following properties are ignored when comparing styles:

- Standard[431];
- StyleName[432] (if RichViewCompareStyleNames[653]=*False* (default)).

If FindSuchStyle cannot find the desired style, it returns -1.

If FindSuchStyleEx cannot find the desired style, it adds a new style to the end of Items, assigns properties of **Style** to this new item, sets its Standard[431]=*False*, returns the index of this new item.

Return value:

Index of the found style; FindSuchStyle returns -1 if not found.

See also:

- other TFontInfos.**Find***** methods;
- TRVStyle.FindTextStyle[407];
- TFontInfo.IsEqual[444];
- TParaInfos.FindSuchStyle[425];
- TRVListInfos.FindSuchStyle[426].

Demos:

- Demos\Delphi\Editors\Editor 2\
- Demos\CBuilder\Editors\Editor 2\

(see procedure TForm1.rveStyleConversion, rve.OnStyleConversion[366] event)

1.4.2.1.4.2 TParaInfos

Collection of paragraph attributes (styles). Class of TRVStyle.ParaStyles[398] property.

Unit RVStyle;

Syntax

```
TCustomRVInfos = class(TCollection)
TParaInfos = class(TCustomRVInfos)
```

Hierarchy

TObject
TPersistent
TCollection
TCustomRVInfos

Main Properties

Items[423] – array of TParaInfo[448].

InvalidItem[423] is returned when accessing item which is out of index range.

Methods for Paragraph Style Search

- FindStyleWithAlignment[424];
- FindSuchStyle, FindSuchStyleEx[425].

See Also

- TFontInfos[418] – class of collection of text attributes (styles);
- TRVListInfos[425] – class of collection of list attributes (i.e. styles of bullets and numbering).

This item is returned when accessing Items[423][ItemNo], if (ItemNo<0) or (ItemNo>=TParaInfos.Count)

```
property InvalidItem: TParaInfo[448];
```

(introduced in version 1.7)

This feature is for debugging, this situation should not occur.

Default value

Default value has all properties Items[423][0], but (LeftIndent[451],SpaceBefore[455],RightIndent[454],SpaceAfter[454]) =(1,1,1,1) and red border[451].

See also

- TFontInfos.InvalidItem[418].

Provides indexed access to the items in the collection of paragraph attributes (styles)

```
property Items[Index: Integer]: TParaInfo[448];
```

This is default property of TParaInfos, so you can write

```
MyRVStyle.ParaStyles[398][i]
```

instead of

```
MyRVStyle.ParaStyles.Items[i]
```

Many methods of components inherited from TCustomRichView[128] have ParaNo parameter, which is an index in TParaInfos.Items collection (more specifically, in TCustomRichView.Style[156].ParaStyles[398].Items).

Example:

```
  MyRichView.AddNL[167]('Hello', 0, 0);
```

Unlike normal collections, when accessing item with invalid Index (for example, Index<0 or Index>=Count), no

exception occurs. InvalidItem[423] is returned instead.

Adds a new style to the collection of paragraph styles.

```
function Add: TParaInfo[448];
```

This method overrides TCollection.Add method to return TParaInfo instance.

Call Add to create a paragraph style in the collection. The new item is placed at the end of the Items[423] array.

Return value:

Created collection item.

See also:

- TParaInfo.Create[455].

Copies properties of this collection to the destination object **Dest**.

```
procedure AssignTo(Dest: TPersistent); override;
```

TCollection AssignTo method is overridden to make it possible to assign names of styles (TParaInfo.StyleName[432]) to TStrings.

Do not call AssignTo directly, this method is called from **Dest**.Assign method.

Example:

```
MyListBox: TListBox;
...
MyListBox.Items.Assign(MyRVStyle.ParaStyles[398]);
```

A constructor, creates and initializes a collection of paragraph styles.

```
constructor Create(Owner: TPersistent);
```

Do not create collection of styles yourself. It is created and destroyed by TRVStyle component as ParaStyles[398] property.

Searches for style having the specified alignment

```
function FindStyleWithAlignment(BaseStyle: Integer;
  Alignment: TRVAlignment): Integer;
```

(Introduced in version 1.5)

Searches for style having all properties of the **BaseStyle**-th style, but Alignment[449]=**Alignment**.

BaseStyle is an index in Items[423].

The following properties are ignored when comparing styles:

- Standard[431];
- StyleName[432] (if RichViewCompareStyleNames[653]=*False* (default)).

Return value:

Index of style or -1, if not found.

See also methods:

- FindSuchStyle[425].

Searches for a style having properties like **Style**.

```
function FindSuchStyle(BaseStyle: Integer; Style: TParaInfo[448];
  Mask: TRVParaInfoProperties[637]): Integer;
function FindSuchStyleEx(BaseStyle: Integer; Style: TParaInfo[448];
  Mask: TRVParaInfoProperties[637]): Integer;
```

(Introduced in version 1.5 and version 14)

The **BaseStyle**-th style is checked first. **BaseStyle** is an index in Items[423].

Mask is a set of properties to compare (other properties are ignored). Use RVAllParaInfoProperties[637] constant to compare all properties.

The following properties are ignored when comparing styles:

- Standard[431];
- StyleName[432] (if RichViewCompareStyleNames[653]=*False* (default)).

If **FindSuchStyle** cannot find the desired style, it returns -1.

If **FindSuchStyleEx** cannot find the desired style, it adds a new style to the end of Items, assigns properties of **Style** to this new item, sets its Standard[431]=*False*, returns the index of this new item.

Return value:

Index of the found style; **FindSuchStyle** returns -1 if not found.

See also methods:

- FindStyleWithAlignment[424];
- TRVStyle.FindParaStyle[407];
- TParaInfo.IsEqual[456];
- TFontInfos.FindSuchStyle[422];
- TRVListInfos.FindSuchStyle[426].

Demos:

- Demos\Delphi\Editors\Editor 2\
- Demos\CBuilder\Editors\Editor 2\

(see procedure TForm1.rveParaStyleConversion, rve.OnParaStyleConversion[364] event)

1.4.2.1.4.3 TRVListInfos

Collection of paragraph list attibutes (bullets and numbering). Class of TRVStyle.ListStyles[396] property.

Unit RVStyle.

Syntax

```
TCustomRVInfos = class(TCollection)
TRVListInfos = class(TCustomRVInfos)
```

Hierarchy

TObject
TPersistent
TCollection
TCustomRVInfos

Main Properties

Items[426] – array of TRVListInfo[459].

Main Methods

- FindSuchStyle[426]
- FindStyleWithLevels[426]

See Also

- TFontInfos 418 – class of collection of text attributes (styles);
- TParaInfos 423 – class of collection of paragraph attributes (styles);
- List markers item type 99.

Provides indexed access to items in the collection of paragraph list attributes (bullets and numbering)

```
property Items[Index:Integer] : TRVListInfo 459;
```

This is default property of TRVListInfos, so you can write

```
MyRVStyle.ListStyles 396[i]
```

instead of

```
MyRVStyle.ListStyles.Items[i]
```

Adds a new style to the collection of paragraph list styles (bullets and numbering)

```
function Add: TRVListInfo 459;
```

This method overrides TCollection.Add method to return TRVListInfo instance.

Call Add to create a new list style in the collection. The new item is placed at the end of the Items 426 array.

Add returns the new collection item.

Copies properties of this collection to the destination object **Dest**.

```
procedure AssignTo(Dest: TPersistent); override;
```

TCollection AssignTo method is overridden to make it possible to assign names of styles (TParaInfo.StyleName 432) to TStrings.

Do not call AssignTo directly, this method is called from **Dest**.Assign method.

Example:

```
MyListBox: TListBox;
...
MyListBox.Items.Assign(MyRVStyle.ListStyles 396);
```

Searches for the style having the specified levels. Adds if not found (optionally)

```
function FindStyleWithLevels(Levels: TRVListLevelCollection;
  const StyleNameForAdding: String; AddIfNotFound: Boolean): Integer;
```

This method searches for the style having levels with properties like in **Levels**.

If **AddIfNotFound**=*True* and style is not found, it adds a new style to the end of collection, assigns **Levels** to its Levels 459 and **StyleNameForAdding** to its StyleName 432. The created style has Standard 431 property = *False*.

Return value:

Index of style, or -1 if not found (and not added).

See also:

- FindSuchStyle 426.

Searches for the style. Adds if not found (optionally)

```
function FindSuchStyle(Style: TRVListInfo 459; AddIfNotFound: Boolean): Integer;
```

This method searches for the style having all properties of **Style**.

If **AddIfNotFound**=*True* and style is not found, it adds a new style to the end of collection and assigns all

properties of **Style** to it. The created style has Standard[431] property = *False*.

The following properties are ignored when comparing styles:

- Standard[431];
- StyleName[432] (if RichViewCompareStyleNames[653]=*False* (default)).

Return value:

Index of style, or -1 if not found (and not added).

See also:

- FindStyleWithLevels[426];
- TFontInfos.FindSuchStyle[422];
- TParaInfos.FindSuchStyle[425].

1.4.2.1.4.4 TRVStyleTemplateCollection

Collection of "real" styles of text and paragraphs. Class of TRVStyle.StyleTemplates[402] property.

Unit RVStyle.

Syntax

```
TRVStyleTemplateCollection = class(TCollection)
```

(introduced in version 14)

Hierarchy

TObject
TPersistent
TCollection

Main Properties

- Items[428] – array of TRVStyleTemplate[460].
- DefStyleName[427] – a format string used to generate names of new items.

Main Methods

- AssignStyleTemplates[428] assigns another collection to this collection;
- FindById, FindItemById[429] search for items by their identifiers;
- FindByName, FindItemByName[429] search for items by their names;
- LoadFromRVST, SaveToRVST[429] save and load the collection to a file;
- ResetNameCounter[430] resets a counter used to generate names for new items.

See also

- Styles and style templates[40].

Specifies the name for new style templates.

```
property DefStyleName: String;
```

This string must contain '%d' as a substring. It will be substituted by a number to generate an unique name. This name will be assigned to Name[461] of new items in this collection.

Default value:

'Style %d'

See also:

- ResetNameCounter[430]

Returns a list of identifiers that cannot be assigned to new items.

```
property ForbiddenIds: TRVIntegerList;
```

This property is useful to implement editing styles (in a user interface) before calling TRichViewEdit.ChangeStyleTemplates[304]. When the user deletes a style template, add its Id[461] to this list. Otherwise, if a new style template accidentally receives the same Id as a deleted style template, text and paragraphs linked to the deleted style template will be linked to the new style template.

Provides indexed access to items in the collection of style templates.

```
property Items[Index:Integer]: TRVStyleTemplate[460];
```

This is default property of TRVStyleTemplateCollection, so you can write

```
MyRVStyle.StyleTemplates[402][i]
```

instead of

```
MyRVStyle.StyleTemplates.Items[i]
```

Returns the item having Name[461]='Normal'.

```
property NormalStyleTemplate: TRVStyleTemplate[460];
```

If such an item does not exist in Items[428], the property returns *nil*.

Copies items from **Source** to this collection.

```
procedure AssignStyleTemplates(Source: TRVStyleTemplateCollection; CopyIds: Boolean);
```

If **CopyIds**=*True*, Id[461] properties of **Source**.Items[428] are copied to new items.

If **CopyIds**=*False*, the method generates new Id for new items, retaining NextId[462] and ParentId[463] relations.

Note:

StyleTemplates1.Assign(StyleTemplates2) is equivalent to StyleTemplates1.AssignStyleTemplates(StyleTemplates2, False).

See also:

- TCustomRichViewEdit.ChangeStyleTemplates[304].

Assign style templates to a string list.

```
type
  TRVStyleTemplateKinds = set of TRVStyleTemplateKind[461];
procedure AssignToStrings(Strings: TStrings; AssignIds: Boolean;
  Kinds: TRVStyleTemplateKinds; ExcludeThisStyle: TRVStyleTemplate[460];
  OnlyPotentialParents: Boolean);
```

Parameters:

Strings – a destination string list. Names[461] of style templates are added as items to this string list.

If **AssignIds**=*True*, the method saves Ids[461] of style templates as Objects of **Strings** (type-casted to TObject).

Only style templates having Kinds[461] belonging to **Kinds** are added.

If **ExcludeThisStyle**<>*nil*, it is not added to **Strings**.

If **OnlyPotentialParents**=*True*, and **ExcludeThisStyle**<>*nil*, only style templates that can be parents of **ExcludeThisStyle** are added to **Strings** (direct or indirect children of **ExcludeThisStyle** are not added).

Note: this method may be useful only if you display Names[461] of all style templates to users (for example, you may implement localization of names of standard styles; in this case, you need to implement a similar function yourself).

The methods reset formatting of the specified text/paragraph attributes (styles).

```
procedure ClearTextFormat(ATextStyle: TCustomRVFontInfo[432]);
procedure ClearParaFormat(AParaStyle: TCustomRVParaInfo[448]);
```

ClearTextFormat clears formatting from **ATextStyle**. The method resets all its properties to default values, then:

- if **ATextStyle** is TFontInfo[444], and **ATextStyle**.Jump[445]=True, and "Hyperlink" style template exists in Items[428], the method applies it to **ATextStyle**;
- otherwise, the method assigns **ATextStyle**.StyleTemplateId = -1.

ClearParaFormat clears formatting from **AParaStyle**. The method resets all its properties to default values, then:

- if "Normal" style template exists in Items[428], the method applies it to **AParaStyle**;
- otherwise, the method assigns **AParaStyle**.StyleTemplateId = -1.

Searches for the item having the specified identifier[461] **Id**.

```
function FindById(Id: TRVStyleTemplateId): Integer;
function FindItemById(Id: TRVStyleTemplateId): TRVStyleTemplate;
```

FindById returns the index of the found item in Items[428], or -1 if not found.

FindItemById returns the the found item, or *nil* if not found.

See also:

- FindByName, FindItemByName[429]

Searches for the item having the specified name[461] **Name**

```
function FindByName(const Name: TRVStyleTemplateName): Integer;
function FindItemByName(const Name: TRVStyleTemplateName): TRVStyleTemplate;
```

FindByName returns the index of the found item in Items[428], or -1 if not found.

FindItemByName returns the the found item, or nil if not found.

See also:

- FindById, FindItemById[429]

The methods load and save style templates to files.

```
function LoadFromRVST(const FileName: String; Units: TRVStyleUnits[644]): Boolean;
function SaveToRVST(const FileName: String; Units: TRVStyleUnits[644]): Boolean;
```

Files of "RichView Styles" format contain style templates. Normally they have "rvst" extension. Files of these formats can be imported and exported in the RichViewActions' style management dialog window (displayed by TrvActionStyleTemplates action).

When saving, specify **Units** used by the style templates:

- for RVStyle.StyleTemplates[402], this is RVStyle.Units[404];
- for rvActionNew.StyleTemplates or rvActionStyleTemplates.StandardStyleTemplates, this is RVAControlPanel.UnitsProgram.

When loading, **Units** are specified similarly. **LoadFromRVST** converts the loaded style templates to the specified **Units**.

Note: after loading to RVStyle.StyleTemplates[402], clear RVStyle.TextStyles[404] and ParaStyles[398] (because the old items may contain references[432] to previous style templates). Then you can add default text and paragraph styles (see the example[464]).

Example:

Demos*\Editors\StyleTemplates demo loads "default.rvst" in "File | New" command.

Resets the counter used to generate unique names for new Items[428].

```
procedure ResetNameCounter;
```

Names are generated using DefStyleName[427] property. By default, new items receive names: 'Style 1', 'Style 2', ..., with the name counter increased: 1, 2...

This method assigns 1 to the name counter, so the next generated name will include a minimal number providing its uniqueness.

1.4.2.1.5 Classes of Properties - Styles

1.4.2.1.5.1 TCustomRVInfo, TCustomRVTextOrParaInfo

These are ancestor classes for all TRichView attributes (styles): text attributes, paragraph attributes, list attributes.

Unit RVStyle.

Syntax

```
TCustomRVInfo = class(TCollectionItem)
TCustomRVFontOrParaInfo = class(TCustomRVInfo)
```

Hierarchy

TObject
TPersistent
TCollectionItem

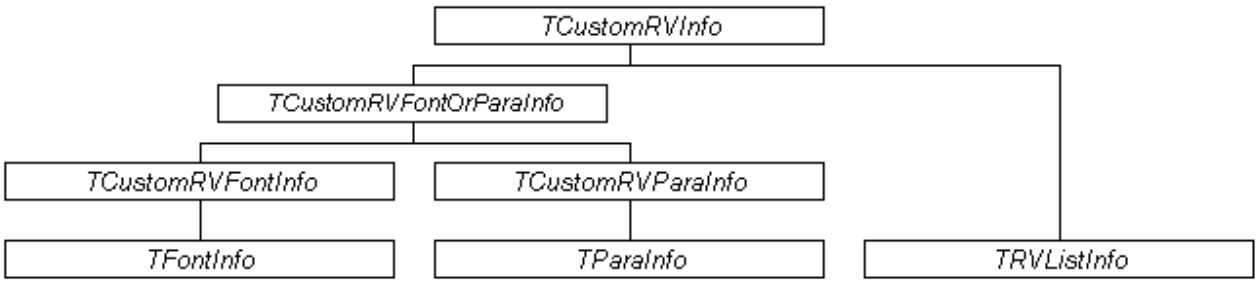
Using

These classes are not used directly. Classes inherited from them are used instead.

The following classes are inherited from TCustomRVInfo (all defined in RVStyle unit).

Class	Meaning
TCustomRVFontInfo[432]	This class introduces the most of properties related to text attributes. Ancestor of TFontInfo.
TCustomRVParaInfo[448]	This class introduces the most of properties related to paragraph attributes. Ancestor of TParaInfo.
TFontInfo[444]	Text style, defines text attributes. Class of items in the collection TRVStyle[385].TextStyles[404].
TParaInfo[456]	Paragraph style, defines paragraph attributes. Class of items in the collection TRVStyle[385].ParaStyles[398].
TRVListInfo[459]	List style (i.e. style of paragraphs' bullets & numbering). It has Levels[459] collection defining attributes of paragraph list markers[99]. Class of items in the collection TRVStyle[385].ListStyles[396].

The class hierarchy is explained on the diagram below.



Properties

TCustomRVInfo introduces the following properties:

- Standard 431;
- StyleName 432.

TCustomRVFontOrParaInfo introduces the following properties:

- StyleTemplateId 432.

Identifies standard (essential) styles

```
property Standard: Boolean;
```

(introduced in version 1.6)

Description

Standard styles are not deleted by DeleteUnusedStyles 181.

This property is ignored when comparing and searching styles.

All styles which are added automatically while loading from RTF or inserting from RVF are not standard.

All styles added by TRVStyle.FindTextStyle 407, TRVStyle.FindParaStyle 407, TFontInfos.FindSuchStyleEx 422, TParaInfos.FindSuchStyleEx 425 are not standard.

Styles which are added automatically while loading from RVF preserve value of this property.

Usage without StyleTemplates 402

If StyleTemplates are not used 158, standard styles may represent "real" styles, while nonstandard styles may represent additional text and paragraph attributes. Initially, you can fill the collection with standard styles, which will never be removed (unless replaced when loading an RVF file). When working with a fixed set of styles 134, it's recommended to make all styles standard. When working with a dynamic set of styles, it is usually enough to have only one standard text and paragraph style initially (all other styles will be added on editing).

Usage with StyleTemplates 402

If StyleTemplates are used, it's recommended to have only one standard paragraph style initially, linked 432 to NormalStyleTemplate 428, and only one standard text style not linked to any style template (but its ParaStyleTemplateId 439 should point to NormalStyleTemplate 428 as well). All other styles will be added on editing.

Default value:

True

See also:

- TFontInfo 444 – class of text attributes (style);
- TParaInfo 448 – class of paragraph attributes (style);
- TRVListInfo 459 – class of paragraph list attributes (style);

- RichViewResetStandardFlag[656] typed constant.

Name of this style.

```
property StyleName: String;
```

This name is used:

- when displaying a design-time editor for collections of styles;
- when assigning text styles to TStrings[419].
- when saving RTF without using StyleTemplates[158] and with *rvrtfSaveStyleSheet*[150] option;

This property is ignored when comparing and searching styles (see also RichViewCompareStyleNames[653] typed constant).

Names of styles do not need to be unique in the collection (unless you save RVF with *rvfoUseStyleNames*[152] option).

This property makes sense if you use a predefined set of styles[134]. If you add styles dynamically, you can ignore this property.

Links this style to a style template.

```
property StyleTemplateId: TRVStyleTemplateId[644];
```

(introduced in version 14)

This property is used only if style templates are used[158].

TCustomRVFontInfo[432] may be linked to style templates having Kind equal to *rvstkParaText* or *rvstkText*.

TCustomRVParaInfo[448] may be linked to style templates having Kind equal to *rvstkParaText* or *rvstkPara*.

See also:

- TCustomRVFontInfo.ParaStyleTemplateId[439]

1.4.2.1.5.2 TCustomRVFontInfo

This is an ancestor class for TFontInfo[444] (class of item in collection of text styles[404]).

TCustomRVFontInfo introduces the most of properties of text.

Unit RVStyle.

Syntax

```
TCustomRVFontInfo = class(TCustomRVFontOrParaInfo[430])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TCustomRVFontOrParaInfo[430]
TCustomRVInfo[430]

Properties

Font attributes (similar to properties of TFont):

- FontName[436] – font name;
- Size[441] and SizeDouble[441] – font size;
- Color[435] – text color;
- Style[441] – text styles (bold, italic, underline, strikeout);

- Charset 434 – font character set (for non-Unicode text styles).

Additional properties affecting text appearance and layout:

- BackColor 433 – text background color;
- UnderlineColor 442 – underline color;
- UnderlineType 442 – underline style (normal, double, dotted, etc.);
- StyleEx 441 – additional text styles (overline, all-caps);
- SubSuperScriptType 441 – normal text/subscript/superscript;
- VShift 443 moves text up or down from its base line;
- CharScale 434 – horizontal character scaling;
- CharSpacing 435 – spacing between characters;
- BiDiMode 433 – default text direction.

Hypertext properties:

- HoverColor 436 – text highlight color;
- HoverBackColor 436 – text background highlight color;
- HoverUnderlineColor 437 – underline highlight color;
- HoverEffects 437 – additional highlight effects (underline);
- JumpCursor 438 – mouse cursor for this text.

Properties related to editing in TCustomRichViewEdit 282:

- Protection 439 – text protection options.

Properties related to text export:

- Options 438 allows to create RTF and HTML codes.

Methods

IsEqual 444 compares this text style with another one.

You can use Assign 443 method to assign:

- TCustomRVFontInfo to TCustomRVFontInfo;
- TFont to TCustomRVFontInfo;
- TCustomRVFontInfo to TFont.

See Also

Classes:

- TFontInfo 444 – class inherited from this class, text attributes (style);
- TFontInfos 418 – collection of text attributes (styles) (TFontInfo 444).

Background color for text of this style

```
property BackColor: TColor;
```

Set BackColor=*clNone* for transparent text background.

Default value:

clNone

See also:

- HoverBackColor 436;
- Color 435;
- HoverColor 436.

See also properties of TRVStyle:

- TextBackgroundKind 402.

Specifies default direction for the text of this style.

```
property BiDiMode: TRVBiDiMode 623;
```

(introduced in version 1.6)

Value	Meaning
rvbdUnspecified	Text direction is inherited from higher level, i.e. from paragraph (see BiDiMode 450 of paragraph style)
rvbdLeftToRight	Default text reading order is left-to-right
rvbdRightToLeft	Default text reading order is right-to-left

Default value:

rvbdUnspecified

See also:

- Bidirectional text in TRichView 69.

Horizontal character scaling value for text of this style, %

```
property CharScale: Integer;
```

(Introduced in version 1.5)

By default, value of this property is equal to 100%, and text of this style is displayed with normal proportions. Set it to larger value to make characters wider, and to lesser value to make them narrower.

This value does not affect the height of text.

Example:

This example shows text formatted with CharScale = 100, 150, and 80.



CharScale Example

Warning: the standard Delphi TFont class does not support character scaling. So be careful if you want to use this value in combination with custom drawing 415. Any changes of RichView's Canvas.TFont will reset font scaling to default.

Default value:

100

Specifies character set for text of this style.

```
property Charset : TFontCharset;
```

Set Charset to identify the character set of the font. Each typeface (specified by the FontName 436 property) supports one or more character sets.

Similar to TFont.Charset.

If Unicode 446=*True*, this Charset is not used when drawing text (but still used inside some methods performing conversion ANSI<->Unicode, for determining the code page (language) of text of this style).

Value of RVStyle.TextStyles 404[0].Charset is used to determine document charset when saving HTML files

(written in <meta> tag).

Default value:

DEFAULT_CHARSET

Specifies intercharacter spacing for text of this style.

```
property CharSpacing: TRVStyleLength;
```

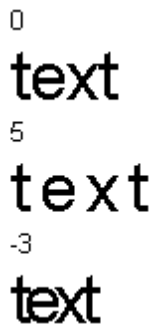
(introduced in version 1.6)

This property defines expansion (if positive) or compression (if negative) of space between characters.

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units of TRVStyle component which owns this text style.

Example:

This example shows text formatted with CharSpacing = 0, 5, and -3 pixels.



CharSpacing Example

Note: if TextEngine is Windows API, all expanded characters are aligned to the left side (space is added to the right side of the characters).

if TextEngine is Uniscribe:

- characters are aligned to the leading side (left side for left-to-right text, right side for right-to-left text)
- kashidas are used for expanded Arabic text
- some characters may be grouped in clusters, a total spacing will be applied to the whole cluster; however, the caret positions are calculated as if there are no clusters.

Default value:

0

Specifies color for text of this style.

```
property Color: TColor;
```

Similar to TFont.Color.

Default value:

clWindowText

See also:

- BackColor;
- HoverColor;
- HoverBackColor.

Specifies font typeface for text of this style.

```
type TFontName = type string;
property FontName: TFontName
```

If the font family described by FontName includes multiple character sets, be sure to set the Charset 434 property as well (if Unicode 446=*False*).

Similar to TFont.Name.

Default value:

'Arial'

Specifies background highlight color for hypertext links of this style.

```
property HoverColor: TColor;
```

(introduced in version 1.2)

If this style is a hypertext style (Jump 445=*True*), background of text of this style can be highlighted with this color when user moves the mouse pointer above it. Set HoverBackColor=BackColor 433 to disable effect.

By default, RichViewEdit 282 also highlights hyperlinks when the caret is moved inside them. This effect can be disabled by including *rvoNoCaretHighlightJumps* in RichViewEdit.EditorOptions 291 property.

This property has no effect for non-hypertext styles.

Default value:

clNone

See also properties:

- BackColor 433;
- HoverColor 436;
- Jump 445;
- JumpCursor 438;
- HoverEffects 437;
- HoverUnderlineColor 437.

See also properties of TRVStyle:

- HoverColor 393;
- FullRedraw 392;
- TextBackgroundKind 402.

Specifies highlight color for hypertext links of this style.

```
property HoverColor: TColor;
```

If this style is a hypertext style (Jump 445=*True*), text of this style can be highlighted with this color when user moves the mouse pointer above it.

If this property is equal to *clNone*, TRVStyle.HoverColor 393 is used. If it is, in its order, equal to *clNone*, no hover color effect is used.

By default, RichViewEdit 282 also highlights hyperlinks when the caret is moved inside them. This effect can be disabled by including *rvoNoCaretHighlightJumps* in RichViewEdit.EditorOptions 291 property.

This property has no effect for non-hypertext styles.

Default value:

clNone

See also properties:

- Color 435;
- HoverBackColor 436;
- Jump 445;

- JumpCursor 438;
- HoverEffects 437;
- HoverUnderlineColor 437.

See also properties of TRVStyle:

- HoverColor 393;
- FullRedraw 392.

Specifies additional effects for hyperlinks.

```
type
  TRVHoverEffect = (rvheUnderline);
  TRVHoverEffects = set of TRVHoverEffect;
property HoverEffects: TRVHoverEffects;
```

(introduced in version 10)

If this style is a hypertext style (Jump 445=*True*), these effects are applied to text of this style when user moves the mouse pointer above it.

By default, RichViewEdit 282 also applies these effects to hyperlinks when the caret is moved inside them. This effect can be disabled by including *rvoNoCaretHighlightJumps* in RichViewEdit.EditorOptions 291 property.

This property has no effect for non-hypertext styles.

Effect	Meaning
rvheUnderline	Text is underlined.

This underline may have a custom color, see HoverUnderlineColor 437 property.

Default value:

[]

See also properties:

- Jump 445;
- HoverColor 436;
- HoverBackColor 436;
- JumpCursor 438.

See also properties of TRVStyle:

- FullRedraw 392.

Specifies underline highlight color for hypertext links of this style.

```
property UnderlineColor: TColor;
```

(introduced in version 10)

Underline is displayed only if *fsUnderline* is included in Style 441, or *rvheUnderline* is included HoverEffects 437.

If this style is a hypertext style (Jump 445=*True*), underline of text of this style can be highlighted with this color when user moves the mouse pointer above it.

If this property is equal to *clNone*, UnderlineColor 442 is used. If it is, in its order, equal to *clNone*, HoverColor 436 is used. If it is, in its order, equal to *clNone*, Color 435 is used.

RTF Note:

This attribute cannot be stored in RTF files.

HTML Note:

Colored underlines are not supported in HTML and CSS.

SaveHTML 216 ignores this attribute. SaveHTMLEx 217 saves colored underline as a bottom border side of the

text's element.

Default value:

clNone

See also:

- UnderlineType 442.

Specifies mouse cursor for hypertext of this style.

```
property JumpCursor: TCursor;
```

This cursor appears when user moves the mouse pointer to text item of this style in RichView. This property has effect only for hypertext items (Jump 445=*True*).

Default value:

crJump (101) (system hypertext cursor or )

See also properties of TRVStyle:

- JumpCursor 394.

See also:

- Cursors used by components of RichView family 77.

Options for exporting text of this style to RTF and HTML.

```
type
  TRVTextOption = (rvteoHTMLCode, rvteoRTFCode, rvteoDocXCode, rvteoDocXInRunCode, rvteoHid
  TRVTextOptions = set of TRVTextOption;

property NextStyleNo: Integer;
```

(introduced in version 1.6)

Option	Meaning
rvteoRTFCode	If set, '{', '}', '\' are saved in RTF as they are. **Example:** set this option, write text of this style: '\animtext1 Viva Las Vegas', copy to RTF and paste in MS Word.
rvteoHTMLCode	If set, '&', '<', '>', and multiple spaces are saved in HTML as they are. **Example:** set this option, write text of this style: '<marquee width=200>Attention. Spaceship is arriving at the Second Terminal.</marquee>', save as HTML and open in web browser.
rvteoDocXCode	If set, text of this style is saved to DocX as it is (converted to UTF-8), in place of <w:r>. No character properties are saved.
rvteoDocXInRunCode	If set, text of this style is saved to DocX as it is (converted to UTF-8) in place of <w:t> (inside <w:r>).
rvteoHidden	If set, this text is hidden. Hidden text is displayed only if *rvoShowHiddenText* is included in RichView.Options 146

All "code" options are mutually exclusive. If two or more different "code" options are included, text of this style is not saved in any of RTF, HTML, DocX.

If the both *rvteoDocXCode* and *rvteoDocXInRunCode* are specified, *rvteoDocXCode* is used.

Default value:

[]

See also:

- TCustomRichView.SaveRTF 219;
- TCustomRichView.SaveDocX 215;
- About HTML export 62.

Links this text style to the style template of paragraph.

```
property ParaStyleTemplateId: TRVStyleTemplateId 644;
```

(introduced in version 14)

This property is used only if style templates are used 158.

This property is redundant: the style template of paragraph is linked using TParaInfo 456.StyleTemplateId 432. However, it is necessary to distinguish text styles having all identical properties but used for text in paragraphs linked to different style templates.

TRichView provides for all text items that their TextStyle.ParaStyleTemplateId=ParaStyle.StyleTemplateId. If not, it automatically links the item to another text style with the correct properties (adding it to the end of TextStyles 404, if necessary).

Protects text of this style from specific operations in editor.

```
property Protection: TRVProtectOptions;

type
  TRVProtectOption =
    (rvprStyleProtect, rvprStyleSplitProtect,
     rvprModifyProtect,
     rvprDeleteProtect, rvprConcateProtect,
     rvprRVFInsertProtect, rvprDoNotAutoSwitch,
     rvprParaStartProtect,
     rvprSticking, rvprSticking2, rvprSticking3,
     rvprStickToTop, rvprStickToBottom);
  TRVProtectOptions = set of TRVProtectOption;
```

Used in RichViewEdit 282. Text of this style has special behavior when editing.

General Protection Options

Option	Meaning
rvprStyleProtect	Protects text from changing style with ApplyTextStyle 300, ApplyTextStyleTemplate 301, ApplyStyleTemplate 299. This protection is ignored by ApplyStyleConversion 299, but you can take it into account manually in OnStyleConversion 366 event.
rvprStyleSplitProtect	Protects from applying text style to the part of text item. ApplyTextStyle 300 and ApplyStyleConversion 299 methods apply new style to the whole item having this protection, even if it is selected only partially.
rvprModifyProtect	Protects text items of this style from modification, but does not protect from deleting as a whole.
rvprDeleteProtect	Protects text items of this style from deletion as a whole, but does not protect from modification.

	See also: rvepDeleteProtect item property 627.
rvprConcateProtect	Protects texts items from concatenation with adjacent item of the same style.
rvprRVFInsertProtect	If set, items of this style will not be inserted when inserting RVF. Affects methods introduced in editor only (**InsertRV***Ed**, pasting).
rvprDoNotAutoSwitch	If set, editor never switches the current text style to this one automatically. If this style was made current by the user (by assigning CurTextStyleNo 290 property), the editor switches back to another (previously used) style after the first performed operation.
rvprParaStartProtect	Disallows moving text item of this style to/from new line; blocks Enter key when the caret is in text of this style.

Sticking

Option	Meaning
rvprSticking	Disallows insertion between two text items having this protection (allows to stick together several text items of different text styles).
rvprSticking2	The same.
rvprSticking3	The same.
rvprStickToTop	Disallows insertion before the text of this style, if this text is a first document item.
rvprStickToBottom	Disallows insertion after the text of this style, if this text is a last document item.

rvprSticking, *rvprSticking2*, and *rvprSticking3* allow to create several groups of stuck items. It's not possible to insert something between items having the same protection option, but it's possible to insert between items having different options, for example *rvprSticking* and *rvprSticking2*. In addition to these options, you can use OnCheckStickingItems 357 event.

Note:

TFontInfo.SingleSymbols property is removed in version 1.3. When loading forms saved with an older version, or when loading styles from ini-files or registry, SingleSymbols=*True* is converted to Protection=[*rvprStyleProtect*, *rvprDoNotAutoSwitch*].

Default value:

[]

See also:

- Options 453 of paragraph style.

Demo projects:

- Demos\Delphi\Assorted\Fields\
- Demos\CBuilder\Assorted\Fields\

The properties specify the font height.

```
property Size: Integer;
property SizeDouble: Integer;
```

(SizeDouble is introduced in version 15)

Size specifies the font size in points [74], similarly to TFont.Size.

SizeDouble specifies the font size in half-points, i.e. its value is twice greater than **Size**.

Assigning to one of these properties changes value another one. **SizeDouble** allows to specify a font size with higher precision.

Default values:

- Size: 10
- SizeDouble: 20

Specifies whether the text of this style is normal, italic, underlined, bold, and so on.

```
type  TFontStyle = (fsBold, fsItalic, fsUnderline, fsStrikeOut);
type  TFontStyles = set of TFontStyle;
property Style: TFontStyles;
```

Similar to TFont.Style.

Default value:

[]

See also properties:

- StyleEx [441];
- HoverEffects [437].

Additional attributes for font of text of given style in RichView or its descendants.

```
type TRVFontStyle = (rvfsOverline, rvfsAllCaps);
type TRVFontStyles = set of TRVFontStyle
property StyleEx: TRVFontStyles;
```

Option	Meaning
rvfsOverline	Vertical line above the text
rvfsAllCaps	All capitals

Note:

All-Caps is introduced in version 1.5. It has the following limitations:

- for Unicode text, capitalization works only under WinNT/2000/XP/Vista;
- for ANSI text capitalization, TRichView always uses the system default language/code page (i.e. the result of AnsiUpperCase).

Default value:

[]

Specifies whether this style is a normal text, a subscript, or a superscript.

```
type
  TRVSubSuperScriptType = (rvsssNormal,  rvsssSubscript,  rvsssSuperScript);
property SubSuperScriptType: TRVSubSuperScriptType;
```

(introduced in version 10)

Subscripts and superscripts are displayed with smaller font than specified in Size[441].

Default value:

rvsssNormal

See also properties:

- VShift[443].

Specifies underline color.

```
property UnderlineColor: TColor;
```

(introduced in version 10)

Underline is displayed only if *fsUnderline* is included in Style[441] (or *rvheUnderline* is included HoverEffects[437], under the mouse pointer).

If UnderlineColor=*clNone*, text color is used for underline (Color[435] in normal state, HoverColor[436] for hyperlinks under the mouse pointer).

Values other than *clNone* define underline color. HoverUnderlineColor[437] property may change hyperlink color under the mouse pointer.

RTF Note:

This attribute can be saved and loaded in RTF files.

HTML Note:

Colored underlines are not supported in HTML and CSS.

SaveHTML[216] ignores this attribute. SaveHTMLEx[217] saves colored underline as a bottom border side of the text's element.

Default value:

clNone

See also:

- UnderlineType[442].

Specifies the style of underline.

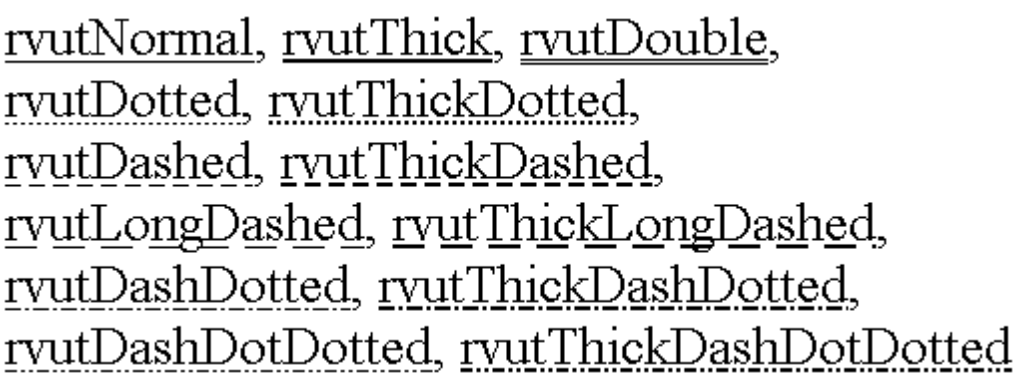
```
type
  TRVUnderlineType = (rvutNormal, rvutThick, rvutDouble,
    rvutDotted, rvutThickDotted,
    rvutDashed, rvutThickDashed,
    rvutLongDashed, rvutThickLongDashed,
    rvutDashDotted, rvutThickDashDotted,
    rvutDashDotDotted, rvutThickDashDotDotted);
property UnderlineType: TRVUnderlineType;
```

(introduced in version 10)

Underline is displayed only if *fsUnderline* is included in Style[441] (or *rvheUnderline* is included HoverEffects[437], under the mouse pointer).

By default, underline color is defined in Color[435] property, but it can be altered by UnderlineColor[442] property (see also HoverColor[436]/HoverUnderlineColor[437] for hyperlink colors under the mouse pointer).

Sample:



UnderlineType Example

RTF Note:

This attribute can be saved and loaded in RTF files.

HTML Note:

Custom underlines are not supported in HTML and CSS.

SaveHTML 216 ignores this attribute. SaveHTMLEx 217 saves custom underline as a bottom border side of the text's element (single, double, dotted and dashed lines are supported).

Default value:

rvutNormal

Vertical offset of text item of this style, in percents of text height.

```
property VShift: Integer;
```

Vertical offset is counted relative to the base line of text. Normal text items have VShift=0.

Use positive values to move text up, negative values to move text down.

This property can be applied not only to normal text, but to subscripts and superscripts 441 (the effect is cumulative).

Default value:

0

See also:

- TRVExtraItemProperty 627 type (*rvepVShift*)

Copies the contents of **Source** to this text style.

```
procedure Assign(Source: TPersistent); override;
```

Call Assign to copy properties of TFont, TCustomRVInfo 430, or TCustomRVFontInfo to this text style.

See also:

- TFontInfo.Assign 446 (TFontInfo 444 is inherited from TCustomRVFontInfo).

Copies the properties of this text style to the destination object **Dest**.

```
procedure AssignTo(Dest: TPersistent);override;
```

Do not call AssignTo directly, this method is called from Assign method of a destination object.

AssignTo allows to assign text style to TFont.

Example:

```
MyEdit.Font.Assign(MyRVStyle.TextStyles 404[0]);
```

A constructor, creates and initializes a new instance of TCustomRVFontInfo.

```
constructor Create(Collection: TCollection);override;
```

The constructor initializes:

```
// Properties inherited from TCustomRVInfo[430]:
Standard[431]     := True;
StyleName[432]    := 'Font Style';
// Properties defined in TCustomRVFontInfo:
Size[441]         := 10;
FontName[436]     := 'Arial';
Color[435]        := clWindowText;
UnderlineColor[442] := clNone;
BackColor[433]    := clNone;
HoverColor[436]   := clNone;
HoverUnderlineColor[437] := clNone;
Charset[434]      := DEFAULT_CHARSET;
JumpCursor[438]   := crDefault;
CharScale[434]    := 100;
```

Other properties are initialized with zeros/*False*/empty values.

Note: usually, when you create a text style in code, you should assign Standard[431] := *False*.

Compares this text style with **Value**.

```
function IsEqual(Value: TCustomRVFontInfo; IgnoreList: TRVFontInfoProperties[633]): Boolean; d
```

(introduced in version 1.3)

Parameters:

Value – object to compare with this text style. The class of this object is TCustomRVFontInfo or inherited from it.

IgnoreList – set of values identifying properties to ignore when comparing, see TRVFontInfoProperties[633].

The following properties are always ignored when comparing styles:

- Standard[431];
- StyleName[432] (if RichViewCompareStyleNames[653]=*False* (default)).

See also:

- TFontInfo.IsEqual[448] (TFontInfo[444] is inherited from this class);
- TCustomRVParaInfo.IsEqual[456].

1.4.2.1.5.3 TFontInfo

This class defines attributes of text items[88] in TRichView[128], TRichViewEdit[282], TDBRichView[373], TDBRichViewEdit[377] controls.

This is an item in the collection of TRVStyle[385].TextStyles[404]. The class of the collection itself is TFontInfos[418].

Unit RVStyle.

Syntax

```
TFontInfo = class(TCustomRVFontInfo[432])
```

Hierarchy

TObject
TPersistent

TCollectionItem
TCustomRVInfo 430
TCustomRVFontOrParaInfo 430
TCustomRVFontInfo 432

Properties

The most of properties are inherited from TCustomRVFontInfo 432.

The new properties are:

- Jump 445 – "is this a hyperlink?"
- Unicode 446 – defines encoding for text of this style (either Unicode (UTF-16) or ANSI (with character set Charset 434);
- NextStyleNo 446 – index of text style to apply after pressing Enter in editor (if style templates are not used 158);
- ModifiedProperties 445 may (temporarily) contain a list of properties having values different from values defined in a style templates (of this text style 432 and paragraph 439).

Methods

IsEqual 448 compares this text style with another one.

You can use Assign 446 method to assign:

- TFontInfo to TFontInfo;
- TCustomRVFontInfo 432 to TFontInfo and vice versa;
- TFont to TFontInfo and vice versa.

See Also

- TFontInfos 418 – class of collection of text styles;
- TParaInfo 448 – class of paragraph style;
- TRVListInfo 459 – class of list style.

Specifies whether this style is a hypertext style.

```
property Jump: Boolean
```

If Jump=*True*, text items of this style are hyperlinks.

The following properties are applied only to hypertext styles:

- HoverColor 436;
- HoverBackColor 436;
- HoverUnderlineColor 437;
- HoverEffects 437;
- JumpCursor 438.

See also:

- RichView hypertext overview 35.

Demo projects:

- Demos\Delphi\Assorted\Hypertext\
- Demos\CBuilder\Assorted\Hypertext\

This property may contain a list of properties not defined in style templates 402 of this style.

```
property ModifiedProperties: TRVFontInfoProperties 633;
```

(introduced in version 14)

This property is used only if style templates are used 158.

This is a redundant property: the difference in properties is determined not by this property, but by the actual difference in values of properties of this style and its style template(s).

This property is ignored when searching for or comparing text styles.

Some methods may change value of this property, so do not use this property to store any information. Value of this property should be calculated just before using.

See also properties:

- StyleTemplateId 432;
- ParaStyleTemplateId 439.

See also properties of TParaInfo:

- ModifiedProperties 457.

See also methods of TRVStyleTemplate:

- UpdateModifiedTextStyleProperties 466;
- ApplyToTextStyle 465 (contains an example).

Defines the text style for following paragraphs.

```
property NextStyleNo: Integer;
```

This property may contain either an index in Style 156.TextStyles 404, or the value -1.

This property is used in RichViewEdit 282, only if UseStyleTemplates 158=*False*. When a user presses Enter key at the end of paragraph, and the current text has this style, a text in a new empty paragraph will be created using this property. If NextStyleNo=-1, a new paragraph will have the same text style as in the current paragraph.

If UseStyleTemplates 158=*True*, this property is ignored. See the the description NextId 462 property of the paragraph's style template 439 about defining text properties for new paragraphs.

Default value:

-1

See also

- TParaInfo.NextParaNo 457.

Specifies encoding for text of this style: Unicode or ANSI.

```
property Unicode: Boolean;
```

(Introduced in version 1.3)

If *False*, text has ANSI encoding, the character set is specified in Charset 434 property. DBCS is not supported.

If *True*, text has Unicode encoding (UTF-16, WideString).

Default value:

For Delphi/C++Builder 2009 or newer: *True*
For older versions: *False*

See also:

- Unicode in RichView 64.

Copies the contents of **Source** to this text style.

```
procedure Assign(Source: TPersistent); override;
```

Call Assign to copy properties of TFont, TCustomRVInfo 430, TCustomRVFontInfo 432, or TFontInfo to this text style.

Example:

```
// Copying TFont to text style:
```

```
MyRVStyle.TextStyles[404][0].Assign(MyEdit.Font);

// Copying text style to text style:
MyRVStyle.TextStyles[404][0].Assign(MyRVStyle.TextStyles[404][1]);
// or, the same:
MyRVStyle.TextStyles[404][0] := MyRVStyle.TextStyles[404][1];
```

A constructor, creates and initializes a new instance of TFontInfo.

```
constructor Create(Collection: TCollection);override;
```

In addition to initializations performed by the inherited constructor[444], this methods initializes:

```
NextStyleNo[446]     := -1;
```

In Delphi/C++Builder 2009 or newer, Unicode[446] is initialized with True.

Other properties introduced in TFontInfo are initialized with zeros/*False*/empty values.

Note: usually, when you create a text style in code, you should assign Standard[431] := *False*.

Draws **s** on **Canvas**.

```
procedure Draw(const s: TRVRawByteString[638]; Canvas: TCanvas; ThisStyleNo: Integer;
  SpaceAtLeft, Left, Top, Width, Height
  {$IFDEF RVUSEBASELINE},BaseLine{$ELSE}{$IFDEF RVHIGHFONTS},DrawOffsY{$ENDIF}{$ENDIF}: I
  TextWidth, SpaceCount: Integer;
  RVStyle: TRVStyle;
  DrawState: TRVTextDrawStates[645];
  Printing, PreviewCorrection, ForMetafile: Boolean;
  ColorMode: TRVColorMode[625]; DefBiDiMode: TRVBiDiMode[623];
  RefCanvas: TCanvas; GraphicInterface: TRVGraphicInterface;
  LeftNoExpIndex, RightNoExpIndex: Integer);
```

Normally, this procedure is called from TRVStyle[385].OnDrawStyleText[415] event. You just need to pass all the parameters of this event to this method. The parameters that do not exist in OnDrawStyleText are marked with "*".

This method assumes that correct settings (font, colors, character spacing etc.) are already applied to **Canvas**.

Parameters:

s – string to draw. If the specified text style is Unicode (see **RVStyle**.TextStyles[404][**ThisStyleNo**].Unicode[446]), this is a raw unicode string. Otherwise, this is an ANSI string (of the character set assigned to **Canvas**.Font.Charset).

Canvas – canvas to paint.

ThisStyleNo must be equal to the index of this text style in **RVStyle**.TextStyles[404].

SpaceAtLeft – distance from **Left** to the beginning of text. It must be 0 (unless you define RVOLDJUSTIFY in RV_Defs.inc).

Left, Top, Width, Height – rectangle where the text must be drawn.

BaseLine – Y coordinate of the text base line. This parameter exists only if RVUSEBASELINE is defined in RV_Defs.inc; in this case, **Canvas**'s text align must include TA_BASELINE.

DrawOffsY: this parameter exists only if RVHIGHFONTS is defined and RVUSEBASELINE is not defined in RV_Defs.inc; in this case, the text top is drawn at **Top+DrawOffsY**.

TextWidth – width of **s**; this parameter is used only if **SpaceCount**<>0. This parameter is ignored, if **PreviewCorrection**=*True* (text width is recalculated before drawing).

SpaceCount:

- if 0, the text is drawn as it is (unless **PreviewCorrection**=*True*);

- if positive, it must be equal to the count of space characters in **s**; in this case, **Width-SpaceAtLeft-TextWidth** is distributed between space characters;
- if equal to -100, **Width-SpaceAtLeft-TextWidth** is distributed between all characters in **s** that can be expanded.

RVStyle* – TRVStyle[385] component containing this text style.

Printing is *True* if the text is drawn while printing or print previewing.

PreviewCorrection is *True* if the text is drawn while print previewing[384], and TRVPrint.PreviewCorrection[516] =*True*. In this case, **s** is fit to **Width-SpaceLeft** by adjusting character spacing. This parameter is ignored, of **Canvas**<>**RefCanvas**.

ForMetafile is *True* if drawing must be compatible with metafiles.

ColorMode allows printing text as black and white (although colors must already be assigned to **Canvas**).

DefBiDiMode is a bi-di mode of paragraph containing this text.

RefCanvas – canvas used to measure text:

- when drawing in TRichView[128], TRichViewEdit[282], TRVReportHelper[503], TRVPrint[479], it must be equal to **Canvas;**
- when drawing in TSRichViewEdit (ScaleRichView), it must be canvas returned by SRichViewEdit.RichViewEdit.RVData.GetScaleRichViewInterface.GetFormatCanvas.

GraphicInterface* – **RVStyle**.GraphicInterface.

LeftNoExpIndex, RightNoExpIndex are used if **SpaceCount**=-100. If not negative, they specify indexes of characters in **s** (from 0) which must not be expanded.

Compares this text style with **Value**.

```
function IsEqual(Value: TCustomRVFontInfo; IgnoreList: TRVFontInfoProperties[633]): Boolean; o
```

(introduced in version 1.3)

Parameter:

Value – object to compare with this text style. The class of this object is TCustomRVFontInfo[432] or inherited from it. If **Value** is TFontInfo, values of NextStyleNo[446] and Unicode[446] properties can be compared as well.

IgnoreList – set of values identifying properties to ignore when comparing.

The following properties are always ignored when comparing styles:

- Standard[431];
- ModifiedProperties[445];
- StyleName[432] (if RichViewCompareStyleNames[653]=*False* (default)).

See also:

- TCustomRVFontInfo.IsEqual[444] (method of the ancestor class);
- TFontInfos.FindSuchStyle[422];
- TParaInfo.IsEqual[458].

1.4.2.1.5.4 TCustomRVParaInfo

This is an ancestor class for TParaInfo[456] (class of item in collection of paragraph styles[398]).

TCustomRVParaInfo introduces the most of properties of paragraph style.

Unit RVStyle.

Syntax

```
TCustomRVParaInfo = class(TCustomRVFontOrParaInfo[430])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TCustomRVFontOrParaInfo 430
TCustomRVInfo 430

Properties

- Alignment, LastLineAlignment 449 – alignment of paragraph (left, right, center, justify or distribute);
- FirstIndent 451 – indent of the first line of paragraph, summarized with LeftIndent;
- LeftIndent 451 – indent from left margin;
- RightIndent 454 – indent from right margin;
- SpaceBefore 455 – spacing above the paragraph;
- SpaceAfter 454 – spacing below the paragraph;
- Border 451 – border around the paragraph;
- Background 450 – background of paragraph;
- Tabs 455 – collection of tab stops;
- LineSpacing 452, LineSpacingType 452 – line spacing;
- BiDiMode 450 – default text direction;
- Options 453 – protection, wrapping.
- OutlineLevel 454 allows defining headings.

Methods

IsEqual 456 compares this paragraph style with another one.

You can use Assign 455 method to assign TCustomRVParaInfo to TCustomRVParaInfo.

See Also

Classes:

- TParaInfo 456 – class inherited from this class, a paragraph style;
- TParaInfos 423 – collection of text styles (TParaInfo 456).

Overview:

- RichView Paragraphs 37

Alignment of paragraph of this style.

```
type
  TRVAlignment =
    (rvaLeft, rvaRight, rvaCenter, rvaJustify, rvaDistribute);
  TRVLastLineAlignment =
    (rvllaDefault, rvllaLeft, rvllaRight, rvllaCenter, rvllaJustify);

property Alignment : TRVAlignment;
property LastLineAlignment: TRVLastLineAlignment;
```

Possible values of **Alignment**:

Value	Meaning
rvaLeft	The paragraph is aligned to the left side.
rvaRight	The paragraph is aligned to the right side.
rvaCenter	The paragraph is centered.

rvaJustify	All lines of the paragraph (except for the last line) are aligned to the both left and right sides by adding space between words (i.e. by increasing widths of space characters). The last line is aligned according to **LastLineAlignment** property.
rvaDistribute	All lines of the paragraph (except for the last line) are aligned to the both left and right sides by adding space between all characters. The last line is aligned according to **LastLineAlignment** property.

LastLineAlignment is used if **Alignment** is equal to *rvaJustify* or *rvaDistribute*. It is applied to the last line of the paragraph (even if the paragraph has the only line). Additionally it is applied to lines that cannot be aligned to the both sides (for example, if a line contains only a single picture).

Possible values of **LastLineAlignment**:

Value	Meaning
rvllaDefault	The line is aligned to the left side if the paragraph is aligned from left to right 450, or to the right side otherwise.
rvllaLeft	The line is aligned to the left side.
rvllaRight	The line is aligned to the right side.
rvllaCenter	The line is centered.
rvllaJustify	The line is aligned according to **Alignment** property.

Note: Microsoft Word 2013 uses the following combinations:

- *rvaJustify* + *rvllaDefault*
- *rvaDistribute* + *rvllaJustify*

Default value:

- Alignment: *rvaLeft*
- LastLineAlignment: *rvllaDefault*

Defines background for paragraphs of this style (transparent by default)

```
property Background:TRVBackgroundRect 466;
```

(Introduced in version 1.2)

See TRVBackgroundRect 466 for additional information.

Note: RTF and HTML support only equal Background.BorderOffsets 466 and Border 451.BorderOffsets 467. RTF and DocX does not support Opacity 467.

See also:

- Paragraph Background 39.

Specifies default text direction for paragraphs of this style.

```
property BiDiMode: TRVBiDiMode 623;
```

(introduced in version 1.6)

Value	Meaning

rvbdUnspecified	Text direction is inherited from higher level, i.e. from TCustomRichView (see BiDiMode 137)
rvbdLeftToRight	Default text reading order is left-to-right
rvbdRightToLeft	Default text reading order is right-to-left

If two adjacent text items have adjacent LTR (RTL) characters, these items will be arranged LTR (RTL), regardless of the paragraph's BiDiMode.

Default value:

rvbdUnspecified

See also:

- Bidirectional text in TRichView 69.

Defines a border around paragraphs of this style (none by default)

```
property Border:TRVBorder 467;
```

(Introduced in version 1.2)

See TRVBorder 467 for additional information.

Note: RTF and HTML support only equal Background 450.BorderOffsets 466 and Border.BorderOffsets 467.

See also:

- Paragraph Background 39.

Indent of the first line for paragraphs of this style.

```
property FirstIndent: TRVStyleLength 644;
```

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units 404 of TRVStyle 385 component which owns this paragraph style.

This value can be negative.

Indent of first paragraph line from the left border (assuming that indents are converted to pixels) = TCustomRichView.LeftMargin 145+LeftIndent 451+FirstIndent.

In paragraphs with bullets or numbering, this property is overridden by FirstIndent 473 of list level.

Default value:

0

See also properties:

- LeftIndent 451, RightIndent 454;
- SpaceBefore 455, SpaceAfter 454.

See also properties of TRVListLevel:

- FirstIndent 473.

Left indent for paragraphs of this style.

```
property LeftIndent: TRVStyleLength 644;
```

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units 404 of TRVStyle 385 component which owns this paragraph style.

Indent for paragraph from left border (assuming that indents are converted to pixels) = TCustomRichView.LeftMargin 145+LeftIndent [+ FirstIndent 451 for the first line]

In paragraphs with bullets or numbering, this property is overridden with LeftIndent 476 of list level (if BiDiMode 450 <>*rvbdRightToLeft*).

Default value:

0

See also properties:

- RightIndent 454;
- SpaceBefore 455, SpaceAfter 454.

See also properties of TRVListLevel:

- LeftIndent 476.

Defines spacing between lines in paragraphs of this style

```
type
  TRVLineSpacingValue = type Integer;
property LineSpacing: TRVLineSpacingValue;
```

(Introduced in version 1.5)

The meaning of the property depends on the value of LineSpacingType 452 property.

If LineSpacingType is *rvlsPercent*, this property defines line height in percent of normal line height (a height of the highest character in a line, heights of pictures and controls are not taken into account). Additional space will be added below the line (including the last paragraph line). For example, LineSpacing=100 means single spacing (no additional space will be added), LineSpacing=200 means double spacing. Values lesser than 50 are ignored.

With other values LineSpacingType, this value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units 404 of TRVStyle 385 component which owns this paragraph style.

If LineSpacingType is *rvlsSpaceBetween*, this value will be added below each line (including the last line in a paragraph). Negative values are ignored.

If LineSpacingType is *rvlsLineHeightAtLeast*, this property defines minimal line height. If the actual line height (including all pictures and controls) is less than this value, additional spacing is added (below and above the line).

If LineSpacingType is *rvlsLineHeightExact*, this property defines line height. If the actual line height (including all pictures and controls) is not equal to this value, additional spacing is added/subtracted (below and above the line).

Default value:

100

Defines type of line spacing in paragraphs of this style.

```
type
  TRVLineSpacingType = (rvlsPercent, rvlsSpaceBetween);
property LineSpacingType: Integer;
```

(Introduced in version 1.5, 11)

Value	Meaning
rvlsPercent	LineSpacing 452 defines spacing in % of text height. Extra spacing is added/subtracted below lines.
rvlsSpaceBetween	LineSpacing 452 is a value which will be added between lines, measured in RVStyle.Units 404. This type of line spacing cannot be exported in RTF and HTML. Extra spacing is added below lines.
rvlsLineHeightAtLeast	LineSpacing 452 defines minimal line height, measured in

	RVStyle.Units[404]. Extra spacing is added above and below lines.
rvlsLineHeightExact	LineSpacing[452] defines line height, measured in RVStyle.Units[404]. Extra spacing is added/subtracted above and below lines.

Default value:

rvlsPercent

Several options for paragraph

```
type
  TRVParaOption =
    (rvpaoNoWrap, rvpaoReadOnly,
      rvpaoStyleProtect, rvpaoDoNotWantReturns,
      rvpaoKeepLinesTogether, rvpaoKeepWithNext,
      rvpaoWidowOrphanControl);
  TRVParaOptions = set of TRVParaOption;

property Options: TRVParaOptions;
```

(Introduced in version 1.5 - 10)

Layout Options

Option	Meaning
rvpaoNoWrap	Disallows automatic word wrapping in paragraphs of this style. If you use no-wrap paragraphs, it's recommended to add *rvoClientTextWidth* in RichView.Options[146]

Editing and Protection Options:

Option	Meaning
rvpaoReadOnly	Disallows editing paragraphs of this style in TRichViewEdit[282]. This option does not protect from adding new paragraph when the user presses Enter at the beginning or at the end of the paragraph. This option does not protect this paragraph from deletion together with the larger selection.
rvpaoStyleProtect	If set, ApplyParaStyle[297], ApplyParaStyleTemplate[298], ApplyStyleTemplate[299] cannot change style of paragraphs of this style. This option does not protect from ApplyParaStyleConversion[298], but you can take it into account in OnParaStyleConversion[364] event.
rvpaoDoNotWantReturns	If set, Enter key will be blocked if the caret is inside paragraph of this style.

Printing Options

Option	Meaning
rvpaoKeepLinesTogether	Paragraph of this style is printed on one page, if possible.
rvpaoKeepWithNext	Paragraph of this style is printed on the same page as the next paragraph, if possible. Additionally, this option works like *rvpaoKeepLinesTogether*
rvpaoWidowOrphanControl	Prevents page breaks after the first line and before the last line in paragraphs of this style.

Default value:

[]

See also:

- TFontInfo.Protection 439;
- TCustomRichView.WordWrap 159.
- TRVTableRow.KeepTogether 577

Defines outline level of paragraph.

```
property OutlineLevel: Integer;
```

(introduced in v12)

0 – no outline level (body text).

Positive values define levels of heading (1 – heading level 1; 2 – heading level 2, and so on).

Default value:

0

Right indent for paragraphs of this style.

```
property RightIndent: TRVStyleLength 644;
```

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units 404 of TRVStyle 385 component which owns this paragraph style.

Indent for paragraph from the right border (assuming that indents are converted to pixels) = TCustomRichView.RightMargin 149+RightIndent.

In paragraphs with bullets or numbering, this property is overridden with LeftIndent 476 of list level, if BiDiMode 450 =*rvbdRightToLeft*.

Default value:

0

See also properties:

- LeftIndent 451;
- SpaceBefore 455, SpaceAfter 454.

See also properties of TRVListLevel:

- LeftIndent 476.

Specifies the size of the spacing below the paragraph.

```
property SpaceAfter: TRVStyleLength 644;
```

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units 404 of TRVStyle 385 component which owns this paragraph style.

This value is greater than or equals to zero (assigning a negative value assigns zero).

Default value:

0

See also properties:

- SpaceBefore[455];
- LeftIndent[451], RightIndent[454].

Specifies the size of the spacing above the paragraph.

```
property SpaceBefore: TRVStyleLength[644];
```

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units[404] of TRVStyle[385] component which owns this paragraph style.

This value is greater than or equals to zero (assigning a negative value assigns zero).

Default value:

0

See also properties:

- SpaceAfter[454];
- LeftIndent[451], RightIndent[454].

Defines tab stops and their properties for paragraphs of this style.

```
property Tabs: TRVTabInfos[469];
```

(introduced in v1.9)

This is a sorted collection of TRVTabInfo[470] (sorted by Position[471] in ascending order).

Default value:

Empty collection.

See also properties of TRVStyle:

- DefTabWidth[388].

Copies the contents of **Source** to this paragraph style.

```
procedure Assign(Source: TPersistent); override;
```

Call Assign to copy properties of TCustomRVInfo[430] or TCustomRVParaInfo to this paragraph style.

See also:

- TParaInfo.Assign[458] (TParaInfo[456] is inherited from TCustomRVParaInfo).

A constructor, creates and initializes a new instance of TCustomRVParaInfo.

```
constructor Create(Collection: TCollection);override;
```

The constructor initializes:

```
// Properties, inherited from TCustomRVInfo[430]:
Standard[431]        := True;
StyleName[432]       := 'Paragraph Style';
// Properties defined in TCustomRVParaInfo:
Alignment[449]       := rvaLeft;
LineSpacingType[452] := 100;
LineSpacing[452]     := rvlsPercent;
```

Other properties are initialized with zeros/*False*/empty values.

Note: usually, when you create a paragraph style in code, you should assign Standard[431] := *False*.

Compares this paragraph style with **Value**.

```
function IsEqual(Value: TCustomRVParaInfo;
  IgnoreList: TRVParaInfoProperties[637]): Boolean; dynamic;
```

(introduced in version 1.5)

Parameters:

Value – object to compare with this paragraph style. The class of this object is TCustomRVParaInfo or inherited from it.

IgnoreList – set of values identifying properties to ignore when comparing, see TRVParaInfoProperties[637].

The following properties are always ignored when comparing styles:

- Standard[431];
- StyleName[432] (if RichViewCompareStyleNames[653]=*False* (default)).

See also:

- TParaInfo.IsEqual[458] (TParaInfo[456] is inherited from this class);
- TCustomRVFontInfo.IsEqual[444].

1.4.2.1.5.5 TParaInfo

This is a paragraph style. It defines attributes of paragraphs[37] in TRichView[128], TRichViewEdit[282], TDBRichView[373], TDBRichViewEdit[377] controls.

This is an item in the collection of TRVStyle[385].ParaStyles[398]. The class of the collection itself is TParaInfos[423].

Unit RVStyle.

Syntax

```
TParaInfo = class(TCustomRVParaInfo[448])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TCustomRVInfo[430]
TCustomRVFontOrParaInfo[430]
TCustomRVParaInfo[448]

Properties

The most of properties are inherited from TCustomRVParaInfo[448].

The new properties are:

- DefStyleNo[457] – default text style for paragraph;
- NextParaNo[457] – index of paragraph style to apply after pressing **Enter** in editor (if style templates are not used[158]);
- ModifiedProperties[457] may (temporarily) contain a list of properties having values different from values defined in a style template[432].

Methods

IsEqual[458] compares this paragraph style with another one.

You can use Assign[458] method to assign:

- TParaInfo to TParaInfo;
- TCustomRVParaInfo[448] to TParaInfo and vice versa.

See Also

- TParaInfos 423 – class of collection of paragraph styles;
- TFontInfo 444 – class of text style;
- TRVListInfo 459 – class of list style;
- RichView Paragraphs 37.

Specifies the default text style for paragraphs of this style.

```
property DefStyleNo: Integer;
```

(introduced in version 1.8)

Deprecated! Use style templates 402 to link text and paragraph properties together.

You can use the rvsDefStyle 663 constant (= MaxInt, a large positive integer value) as a StyleNo parameter of AddNL 167 and other methods for adding text.

It means using the paragraph's default text style, i.e. DefStyleNo property (or the 0-th style, if DefStyleNo=-1).

GetItemStyle 194 method and CurTextStyleNo 290 return a translated text style (they never return rvsDefStyle).

Use GetItem 190(i).StyleNo and ActualCurTextStyleNo 288 to get the text style as is.

Default value:

-1

This property may contain a list of properties not defined in the style template 402 of this style.

```
property ModifiedProperties: TRVParaInfoProperties 637;
```

(introduced in version 14)

This property is used only if style templates are used 158.

This is a redundant property: the difference in properties is determined not by this property, but by the actual difference in values of properties of this style and its style template.

This property is ignored when searching for or comparing paragraph styles.

Some methods may change value of this property, so do not use this property to store any information. Value of this property should be calculated just before using.

See also properties:

- StyleTemplateId 432.

See also properties of TFontInfo:

- ModifiedProperties 445.

See also methods of TRVStyleTemplate:

- UpdateModifiedParaStyleProperties 466;
- ApplyToParaStyle 464 (contains an example).

Defines the paragraph style for following paragraphs.

```
property NextParaNo: Integer;
```

(Introduced in version 1.4)

This property may contain either an index in Style 156.ParaStyles 398, or the value -1.

This property is used in RichViewEdit 282, only if UseStyleTemplates 158=*False*. When a user presses Enter key at the end of a paragraph of this style, a new empty paragraph will be created using this property. If NextParaNo=-1 then this new line will have the same paragraph style as the current paragraph.

If UseStyleTemplates[158]=*True*, this property is ignored, NextId[462] property of the paragraph's style template[432] is used instead.

Default value:

-1

See also:

- TFontInfo.NextStyleNo[446].

Copies the contents of **Source** to this paragraph style.

```
procedure Assign(Source: TPersistent); override;
```

Call Assign to copy properties of TCustomRVInfo[430], TCustomRVParaInfo[448], or TParaInfo to this paragraph style.

Example:

```
// Copying paragraph style to paragraph style:
MyRVStyle.ParaStyles[398][0].Assign(MyRVStyle.ParaStyles[398][1]);
// or, the same:
MyRVStyle.ParaStyles[398][0] := MyRVStyle.ParaStyles[398][1];
```

A constructor, creates and initializes a new instance of TParaInfo.

```
constructor Create(Collection: TCollection);override;
```

In addition to initializations performed by the inherited constructor[455], this methods initializes:

```
DefStyleNo[457] := -1;
NextParaNo[457] := -1;
```

Note: usually, when you create a paragraph style in code, you should assign Standard[431] := *False*.

Compares this paragraph style with **Value**.

```
function IsEqual(Value: TCustomRVParaInfo;
  IgnoreList: TRVParaInfoProperties[637]): Boolean; override;
```

(introduced in version 1.5)

Parameters:

Value – object to compare with this paragraph style. The class of this object is TCustomRVParaInfo or inherited from it.

IgnoreList – set of values identifying properties to ignore when comparing, see TRVParaInfoProperties[637].

The following properties are always ignored when comparing styles:

- Standard[431];
- StyleName[432] (if RichViewCompareStyleNames[653]=*False* (default));
- ModifiedProperties[457].

See also:

- TCustomRVParaInfo.IsEqual[456] (method of the ancestor class);
- TParaInfos.FindSuchStyle[425];
- TFontInfo.IsEqual[448].

1.4.2.1.5.6 TRVListInfo

This is a paragraph list style. It defines attributes of list markers [99] in TRichView [128], TRichViewEdit [282], TDBRichView [373], TDBRichViewEdit [377] controls.

This is an item in the collection of TRVStyle [385].ListStyles [396]. The class of the collection itself is TRVListInfos [425].

Unit RVStyle.

Syntax

```
TRVListInfo = class(TCustomRVInfo [430])
```

Hierarchy

TObject
TPersistent
TCollectionItem
TCustomRVInfo [430]

Properties

- Levels [459] – collection of list levels (TRVListLevel [472])
- OneLevelPreview [459] suggests appearance of thumbnail.

Other properties are inherited from TCustomRVInfo [430].

Methods

- AllNumbered [459] returns *True* if all levels represent numbered lists;
- HasNumbering [460] returns *True* if at least one of list levels represents a numbered list.

See Also

- TRVListInfos [425] – class of collection of paragraph list styles;
- TFontInfo [444] – class of text style;
- TParaInfo [448] – class of paragraph style;
- List markers item type [99].

Levels of paragraph list style, collection of TRVListLevel [472] objects.

```
property Levels: TRVListLevelCollection [471];
```

Each level defines properties for bullet or numbering of paragraphs.

Can be used by application to define appearance of preview thumbnail for this list style.

```
property OneLevelPreview: Boolean
```

This property is not used by the components themselves.

If application creates thumbnails in dialog for selecting list style, this property can be used to define thumbnail image.

If it is *True*, only one level of this list should be shown on the thumbnail.

See also:

- Levels [459].

Returns *True* if all levels [459] represent numbered lists

```
function AllNumbered: Boolean;
```

See also:

- HasNumbering [460].

Returns *True* if at least one of list levels[459] represents a numbered list.

```
function HasNumbering: Boolean;
```

See also:

- AllNumbered[459].

1.4.2.1.5.7 TRVStyleTemplate

This class represents a "real" style controlling properties of TFontInfo[444] and TParaInfo[444] objects.

This is an item in the collection of TRVStyle[385].StyleTemplates[402]. The class of the collection itself is TRVStyleTemplateCollection[427].

Unit RVStyle.

Syntax

```
TRVStyleTemplate = class(TCollectionItem)
```

(introduced in version 14)

Hierarchy

TObject
TPersistent
TCollectionItem

Main Properties

- Kind[461] – kind of this style templates (defines it applicability to text or paragraphs);
- Name[461] – unique name;
- Id[461] – unique identifier;
- TextStyle[463], ValidTextProperties[464] – text properties;
- ParaStyle[463], ValidParaProperties[463] – paragraph properties;
- ParentId[463] – a reference to a parent style template;
- NextId[462] – a reference to a style template for following paragraphs;
- QuickAccess[463] – recommends to provide a quick access to this style template in applications UI.

Main Methods

- UpdateModifiedTextStyleProperties[466] fills the set of properties of the specified text style that were changed after applying this style template;
- UpdateModifiedParaStyleProperties[466] fills the set of properties of the specified paragraph style that were changed after applying this style template;
- ApplyToTextStyle[465] applies this style template to the specified text style;
- ApplyToParaStyle[464] applies this style template to the specified paragraph style.

Linking

Each style template has two properties having unique values: Id[461] and Name[461]. Id is read-only, is assigned automatically, the collection provides its uniqueness. Name is assigned by the user, you (the programmer) must provide its uniqueness.

Links[432] from TextStyles[404], ParaStyles[398], and between StyleTemplates are made via Id property.

Name is used when merging style templates of two different documents.

Unlike items in TRVStyle.TextStyles[404] and TRVStyle.ParaStyles[398], style templates are never referred by their index in the collection (except for indices returned after searching by a name or an identifier, and the editor methods for applying to the selection[299]).

See also

- Styles and style templates[40].

An identifier of this style template.

```
property Id: TRVStyleTemplateId[644];
```

Value of this property is generated automatically using a random number generator. Each item has an identifier that is unique in the collection[427].

Different style template collections may have equal values of identifiers of some items by a coincidence, however it does not mean that these style templates are related (unless the collection was duplicated using AssignStyleTemplates[428] with CopyIds=*True*).

This property is used to refer to this style template within a single TRVStyle[385] component:

- another style template may refer to this style-template using ParentId[463] property;
- another style template may refer to this style-template using NextId[462] property;
- text[404] and paragraph[398] styles of the same TRVStyle may refer to this style template using StyleTemplateId[432] property.

Why do style templates need two unique properties, **Id** and Name[461]? There are the following reasons:

- editing style templates in a dialog window: if a user renames a style template, all references to this style template must stay unchanged;
- working with integer values is more efficient than with strings.

This property is not copied when assigning one style template to another.

See also methods of TRVStyleTemplateCollection[427]:

- FindById, FindItemById[429].

Specifies the kind of this style template.

```
type
  TRVStyleTemplateKind = (rvstkParaText, rvstkPara, rvstkText);
property Kind: TRVStyleTemplateKind;
```

Kind	Meaning
rvstkParaText	This style template can be applied both to text and paragraph styles.
rvstkPara	This style template can be applied to paragraphs styles[398] (i.e. paragraph styles can be linked[432] to this style template)
rvstkParaText	This style template can be applied to text styles[404] (i.e. text styles can be linked[432] to this style template)

Assigning a new value to Name[461] may change Kind.

Default value:

rvstkParaText.

Name of the style template.

```
property Name: TRVStyleTemplateName;
```

You must provide that each item in the collection has an unique name, otherwise style templates will be saved to RTF and RVF incorrectly.

Special names:

- Assigning 'Normal' changes Kind[461] to *rvstkPara*.
- Assigning 'heading 1'...'heading 9' changes Kind to *rvstkParaText*.
- Assigning 'Hyperlink' changes Kind to *rvstkText*.

Names are used when merging style templates from different documents.

Some names may be reserved for standard style templates. Actually, there is no strict definition of a "standard

style template". A "standard style template" is a style template having a special meaning for your application. For example, your application may have a command "Insert Abracadabra" inserting some text and formatting using the style template named "Abracadabra Text". In this case, "Abracadabra Text" is a standard style template for your application.

However, there are style templates having a special meaning for TRichView: "Normal" and "Hyperlink".

"Normal" is a paragraph style template. This style template is special, because it uses some "magic". All paragraph style templates are applied to paragraphs and cannot be applied to text. But applying "Normal" to text clears this text format.

The format clearing command does the following. For paragraphs, it assigns the style template "Normal" and clears all additional paragraph attributes. For non-hypertext text, it removes its style template and clears all additional text attributes. For hypertext links, it assigns the style template "Hyperlink" and clears all additional text attributes.

 RichViewActions reserve some other names for standard style templates, but only the following are used specially, and only in ScaleRichView:

- "header" to apply to headers of new documents;
- "footer" to apply to footers of new documents;
- "endnote reference"/"footnote reference" to apply to new footnote[104]/endnote[103] characters, and to note references[108];
- "endnote text", "footnote text" to apply to text in Document[586] of new footnote/endnote.

See also members of TRVStyleTemplateCollection[427]:

- FindByName, FindItemByName[429];
- DefStyleName[427].

Defines the style template to be used in the next paragraph after the paragraph marked by this style template (when a user presses Enter)

```
property NextId: TRVStyleTemplateId[644];
property Next: TRVStyleTemplate; // read-only
```

This property is used only for style templates applied to paragraphs, so it is ignored if Kind[461]=*rvstkText*.

To make some style template a next style template, assign its Id[461] to **NextId** of this style template.

NextId equals to Id[461] of the next style (If **NextId**=-1, the next style-template is this style template itself)

Next returns the next style (or *nil* if **NextId**=-1), i.e. **Next**.Id[461]=**NextId**.

When Enter is pressed, and style templates are used[158]:

- the new paragraph is linked to **Next** (or to this style template, if **Next**=*nil*); its attributes are changed accordingly;
- if the current text is a hyperlink (i.e. its Jump[445]=*True*), the new text is not linked to any style template, its attributes are changed accordingly, its Jump[445]=*False*;
- if the current text is not a hyperlink, the new text is linked to the same style template as the current text.

New TRVStyle.TextStyles[404] and ParaStyles[398] are added, if necessary.

These properties are copied when assigning one style template to another, only if the source and target style templates belong to the same collection.

Default value:

NextId: -1

Paragraph properties.

```
type
  TRVSTParaInfo = class (TCustomRVParaInfo 448);
property ParaStyle: TRVSTParaInfo;
```

ParaStyle is ignored if Kind 461=*rvstkText*

Only sub-properties listed in ValidParaProperties 463 are taken into account. Values of other sub-properties are ignored.

Defines the style template the current style template is based on.

```
property ParentId: TRVStyleTemplateId 644;
property Parent: TRVStyleTemplate; // read-only
```

To make some style template a parent of this style template, assign its Id 461 to **ParentId** of this style template.

To make this style template a root style template, assign -1.

Parent returns the parent style template (or *nil* if **ParentId**=-1), i.e. **Parent**.Id 461=**ParentId**.

If Kind 461=*rvstkText*, only style templates having Kind 461=*rvstkText* can be its parents.

If Kind 461<>*rvstkText*, only style templates having Kind 461<>*rvstkText* can be its parents.

Circular references are not allowed.

The style template inherits (from parents) properties not listed in its ValidTextProperties 464 and ValidParaProperties 463.

These properties are copied when assigning one style template to another, only if the source and target style templates belong to the same collection.

Default value:

ParentId: -1

Specifies that this style template should be readily available for a user via the application's user interface.

```
property QuickAccess: Boolean;
```

For example, when RichViewActions show style templates in TRVStyleTemplateComboBox/ TRVStyleTemplateListBox, quick-access styles are always shown, even if they are not used in the document.

Default value

True

Text properties.

```
type
  TRVSTFontInfo = class (TCustomRVFontInfo 432);
property TextStyle: TRVSTFontInfo;
```

Only sub-properties listed in ValidTextProperties 464 are taken into account. Values of other sub-properties are ignored.

A set of properties of ParaStyle 463 defined in this style template.

```
property ValidParaProperties: TRVParaInfoProperties 637;
```

Only properties included in this set are used. Values of other ParaStyle 463's properties are ignored.

This property is used only if Kind 461<>*rvstkText* (otherwise, none of paragraph properties are used).

Although this property is public, not published, it may be defined at design-time in the Object Inspector (where it is displayed as ValidParaPropertiesEditor). It was made because of a Delphi limitation on the count of values in

published set properties.

See also:

- ParentId[463].

A set of properties of TextStyle[463] defined in this style template.

```
property ValidTextProperties: TRVFontInfoProperties[633];
```

Only properties included in this set are used. Values of other TextStyle[463]'s properties are ignored.

Although this property is public, not published, it may be defined at design-time in the Object Inspector (where it is displayed as ValidTextPropertiesEditor). It was made because of a Delphi limitation on the count of values in published set properties.

See also:

- ParentId[463].

Changes properties of **AParaStyle** according to the properties of this style template.

```
procedure ApplyToParaStyle(AParaStyle: TCustomRVParaInfo[448]);
```

The method changes only properties of **AParaStyle** not listed in **AParaStyle**.ModifiedProperties[457]. If you want to change all properties, assign **AParaStyle**.ModifiedProperties[457]=[] before calling this method. If you want to change only properties controlled by style templates, call UpdateModifiedParaStyleProperties[466] before calling this method.

Applying a style template means applying properties of its ParaStyle[463] listed in ValidParaProperties[463], and properties inherited from Parent[463].

Kind[461] of this style template must be either *rvstkPara* or *rvstkParaText*.

The sequence of operations:

1. The method applies this style template (and its parents) to **AParaStyle**. The method assigns **AParaStyle**.StyleTemplateId[432] = Id[461]. This method may be called for *nil*-objects; in this case, it just assigns **AParaStyle**.StyleTemplateId[432] = -1.
2. Then it resets remaining properties of **AParaStyle** to default values.

Example: resetting TextStyles and ParaStyles for using in new documents, using 'Normal' style template.

```
var StyleTemplate: TRVStyleTemplate[460];
begin
  RVStyle.ListStyles[396].Clear;
  RVStyle.TextStyles[404].Clear;
  RVStyle.TextStyles.Add;
  RVStyle.ParaStyles[398].Clear;
  RVStyle.ParaStyles.Add;
  StyleTemplate := RVStyle.StyleTemplates[402].NormalStyleTemplate[428];
  if StyleTemplate<>nil then begin
    RVStyle.ParaStyles[0].ModifiedProperties[457] := [];
    StyleTemplate.ApplyToParaStyle(RVStyle.ParaStyles[0]);
    RVStyle.TextStyles[0].ModifiedProperties[445] := [];
    TRVStyleTemplate(nil).ApplyToTextStyle[465](RVStyle.TextStyles[0], StyleTemplate);
    RVStyle.TextStyles[0].ParaStyleTemplateId[439] := StyleTemplate.Id[461];
  end;
end;
```

See also:

- ApplyToTextStyle 465

Changes properties of **ATextStyle** according to the properties of this style template.

```
procedure ApplyToTextStyle(ATextStyle: TCustomRVFontInfo 432;
  ParaStyleTemplate: TRVStyleTemplate);
```

The method changes only properties of **ATextStyle** not listed in **ATextStyle**.ModifiedProperties 445. If you want to change all properties, assign **ATextStyle**.ModifiedProperties 445=[] before calling this method. If you want to change only properties controlled by style templates, call UpdateModifiedTextStyleProperties 466 before calling this method.

Applying a style template means applying properties of its TextStyle 463 listed in ValidTextProperties 464, and properties inherited from Parent 463.

Kind 461 of this style template must be either *rvstkText* or *rvstkParaText*.

Up to two style templates can be applied to **ATextStyle**: this style template and **ParaStyleTemplate**. This style template has a higher priority than **ParaStyleTemplate**: if some property is defined both in this style template and in **ParaStyleTemplate**, properties of this style template are applied. Both this style template and **ParaStyleTemplate** may be *nil*.

The sequence of operations:

1. The method applies this style template (and its parents) to **ATextStyle**. The method assigns **ATextStyle**.StyleTemplateId 432 = Id 461. This method may be called for *nil*-objects; in this case, it just assigns **ATextStyle**.StyleTemplateId 432 = -1.
2. Then it applies **ParaStyleTemplate** (if not *nil*) to remaining properties. The method does NOT assign **ATextStyle**.ParaStyleTemplateId 439.
3. Then it resets remaining properties of **ATextStyle** to default values.

Example: inserting a hyperlink in rve:TRichViewEdit 282. If possible, this hyperlink is formatted using 'Hyperlink' style template.

```
var
  OldStyleTemplate, NewStyleTemplate, ParaStyleTemplate: TRVStyleTemplate 460;
  TextStyle: TFontInfo 444;
begin
  // making a copy of the current text attributes (of the current text style)
  TextStyle := TFontInfo 444.Create(nil);
  TextStyle.Assign(rve.Style 156.TextStyles 404[rve.CurTextStyleNo 290]);
  // searching for 'Hyperlink' style template
  NewStyleTemplate := rve.Style.StyleTemplates 402.FindItemByName 429('Hyperlink');
  if NewStyleTemplate<>nil then begin
    // 'Hyperlink' style template exists. applying it to TextStyle
    OldStyleTemplate := rve.Style.StyleTemplates.FindItemById 429(TextStyle.StyleTemplateId
    ParaStyleTemplate := rve.Style.StyleTemplates.FindItemById(TextStyle.ParaStyleTemplat
    OldStyleTemplate.UpdateModifiedTextStyleProperties 466(TextStyle, ParaStyleTemplate);
    NewStyleTemplate.ApplyToTextStyle(FontInfo, ParaStyleTemplate);
    end
  else begin
    // 'Hyperlink' style template does not exists. applying default hyperlink properties
    TextStyle.Color 435 := clBlue;
    TextStyle.Style 441 := [fsUnderline];
  end;
  TextStyle.Jump 445 := True;
  // inserting
  rve.CurTextStyleNo := rve.Style.FindTextStyle 407(TextStyle);
  rve.InsertStringTag 326('TRichView.com', 'http://www.trichview.com');
  // cleaning up
  TextStyle.Free;
```

```
end;
```

See also:

- ApplyToParaStyle 464 (contains one more example)

The methods fill the list of properties having different values in **ATextStyle/AParaStyle** and in this style template.

```
procedure UpdateModifiedTextStyleProperties(ATextStyle: TFontInfo 444;
  ParaStyleTemplate: TRVStyleTemplate);
procedure UpdateModifiedParaStyleProperties(AParaStyle: TParaInfo 456);
```

UpdateModifiedTextStyleProperties compares values of properties of **ATextStyle** and properties of this style template (i.e. properties of TextStyle 463 listed in ValidTextProperties 464 and properties inherited from Parent 463). Properties undefined in this style template (and its parents) are compared with properties of **ParaStyleTemplate** (if **ParaStyleTemplate**<>*nil*). Properties having different values are added in **ATextStyle**.ModifiedProperties 445.

UpdateModifiedParaStyleProperties compares values of properties of **AParaStyle** and properties of this style template (i.e. properties of ParaStyle 463 listed in ValidParaProperties 463 and properties inherited from Parent 463). Properties having different values are added in **AParaStyle**.ModifiedProperties 457.

These methods may be called for *nil*- style templates. In this case, properties of **ATextStyle/AParaStyle** are compared with default values.

The results are used in ApplyToTextStyle 465 and ApplyToParaStyle 464 methods.

1.4.2.1.6 Classes of Properties of Paragraph Style

1.4.2.1.6.1 TRVBackgroundRect

This class defines background of paragraph (colored rectangle). This is a class of Background 450 property of paragraph style.

Unit RVStyle.

Syntax

```
TRVBackgroundRect = class (TPersistent)
```

(Introduced in version 1.2)

Properties:

- BorderOffsets 466;
- Color 467;
- Opacity 467.

See also:

- TCustomRVParaInfo.Background 450;
- Paragraph Borders and Background 39.
- TRVBoxBackgroundRect 594 (background of a floating box)

Distance between the paragraph content and the border of the colored area.

```
property BorderOffsets: TRVRect 615;
```

Sides of this rectangle are measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units 404 of TRVStyle 385 component which owns the paragraph style.

Lager positive values increase padding. Negative values force border to be inside the paragraph content.

Warning:

- Background 450.BorderOffsets.Top must not exceed SpaceBefore 455 property of the same paragraph.
- Background.BorderOffsets.Bottom must not exceed SpaceAfter 454 property of the same paragraph.

- Background.BorderOffsets.Left should not exceed LeftIndent 451 property of the same paragraph.
- Background.BorderOffsets.Right should not exceed RightIndent 454 property of the same paragraph.

Default value:

0,0,0,0 (border is drawn around the paragraph without gaps)

See also:

- Paragraph Background 39

Color of paragraph background.

```
property Color: TColor
```

Default value:

clNone (transparent background)

Opacity of paragraph background

```
property Opacity: TRVOpacity 637
```

(introduced in version 16)

This property is used if Color 467 <> *clNone*.

Default value:

100000 (i.e. 100%)

1.4.2.1.6.2 TRVBorder

This class defines border around paragraph. This is a class of Border 451 property of paragraph style.

Unit RVStyle.

Syntax

```
TRVBorder = class (TPersistent)
```

(Introduced in version 1.2)

Properties:

- BorderOffsets 467 – distance between paragraph content and border;
- Color 468 – border color;
- InternalWidth 468 – distance between border lines (for double and triple borders);
- Style 468 – border style;
- VisibleBorders 468 hides/shows border sides;
- Width 468 – line width.

See also:

- TCustomRVParaInfo.Border 451;
- Paragraph Borders and Background 39.

Spacing between paragraph content and border.

```
property BorderOffsets: TRVRect 615;
```

Sides of this rectangle are measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units 404 of TRVStyle 385 component which owns the paragraph style.

Lager positive values increase spacing. Negative values force the border to be inside the paragraph content.

Warning:

- Border 451.BorderOffsets.Top must not exceed SpaceBefore 455 property of the same paragraph.
- Border.BorderOffsets.Bottom must not exceed SpaceAfter 454 property of the same paragraph.
- Border.BorderOffsets.Left should not exceed LeftIndent 451 property of the same paragraph.

- Border.BorderOffsets.Right should not exceed RightIndent[454] property of the same paragraph.

Default value:

0,0,0,0

See also:

- Paragraph Borders[39].

Color of border

```
property Color: TColor
```

This property cannot be used to hide the border. To hide border, set Style[468]=*rvbsNone*.

Default value:

clWindowText

Spacing between lines in double and triple borders.

```
property InternalWidth: TRVStyleLength[644];
```

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units[404] of TRVStyle[385] component which owns this paragraph style.

For example, if Style[468]=*rvbDouble*, InternalWidth=1, and Width[468]=1, and RVStyle.Units[404]=*rvstuPixels*, you'll see two lines having 1-pixel width, with 1-pixel spacing between them.

Default value:

1

Note: this default value assumes that RVStyle.Units[404]=*rvstuPixels*, otherwise it could be too small.

Style of a border

```
property Style: TRVBorderStyle[623];
```

See TRVBorderStyle[623] for possible values.

Default value:

rvbNone (no border)

Defines which sides of border are visible.

```
property VisibleBorders: TRVBooleanRect[611];
```

Default value:

True, *True*, *True*, *True* (all sides are visible, but border is still invisible because its Style[468]=*rvbNone* by default)

Width of border lines.

```
property Width: TRVStyleLength[644];
```

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units[404] of TRVStyle[385] component which owns this paragraph style.

It's not necessary a total width of a border, It's a width of lines in the border. This property is equal to a width of border only if Style[468]=*rvbSingle*.

Thick lines (Style=*rvbThickInside* or *rvbThickOutside*) have double width.

Default value:

1

Note: this default value assumes that RVStyle.Units[404]=*rvstuPixels*, otherwise it could be too small.

1.4.2.1.6.3 TRVTabInfos

Collection of tab stops (TRVTabInfo[470]) and their properties.

This is a type of TRVStyle[385].ParaStyles[398][i].Tabs[455] property.

Unit RVStyle.

Syntax

```
TRVTabInfos = class(TCollection)
```

(introduced in version 1.9)

Hierarchy

TObject
TPersistent
TCollection

Using

This collection is sorted in ascending order by Position[471] properties of its items.

When you add a new tab stop (using Add method), it is added to the end of the collection. When you assign its Position[471], it is moved to the proper place.

Main properties:

- Items[469];
- Count.

Provides indexed access to the items in the collection of tab stops for the paragraph.

```
property Items[Index:Integer]: TRVTabInfo[470];
```

This is the default property of TRVTabInfos, so you can write

```
MyRVStyle.ParaStyles[398][i].Tabs[455][j]
```

instead of

```
MyRVStyle.ParaStyles[i].Tabs.Items[j].
```

Adds tab stops from **Source**.

```
procedure AddFrom(Source: TRVTabInfos);
```

New tab stops are inserted, existing tab stops are updated.

(tab stops are compared by their Position[471] property)

The collection remains sorted.

Deletes tab stops at the specified **Positions**

```
procedure DeleteList(Positions: TRVIntegerList[614]);
```

All tab stops having Position[471] property equal to one of values in **Positions** are deleted.

Returns the index of tab stop with the given **Position** (or -1 if not found).

```
function Find(Position: Integer): Integer;
```

This function compares value of **Position** parameter with value of Position[471] property of tab stops.

Deletes all tab stops that do not present in **Value**. Only tab stops with all common properties are not deleted.

```
procedure Intersect(Value: TRVTabInfos[469]);
```

1.4.2.1.6.4 TRVTabInfo

Defines properties for tab stop. This is a type of item in TRVStyle[385].ParaStyles[398][i].Tabs[455] collection.

Unit RVStyle.

Syntax

```
TRVTabInfo = class(TCollectionItem)
```

(introduced in version 1.9)

Hierarchy

TObject
TPersistent
TCollectionItem

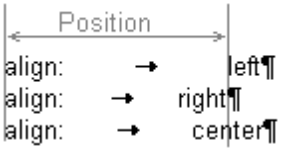
Properties

Position[471] – distance of this tab stop from LeftMargin[145];
Align[470] – alignment of tab relating to the followed text;
Leader[470] – text to fill this tabulator.

Defines tab stop alignment (relative to the text after this tab stop)

```
type
  TRVTabAlign = ( rvtaLeft, rvtaRight, rvtaCenter );

property Align: TRVTabAlign;
```

Image:



Alignments are inverted for right-to-left paragraphs[69].

Default value:

rvtaLeft

See also:

- Position[471].

Repeated text to fill the tabulator space.

```
property Leader: String;
```

Only the following values of this property can be saved in RTF

Character	Description
.	dot

-	hyphen
_	underscore character
·	#$B7, middle dot
=	equals sign

Default value:

" (empty string)

Defines distance of this tab stop from LeftMargin[145].

```
property Position: TRVStyleLength[644];
```

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units[404] of TRVStyle[385] component which owns the paragraph style with this tab stop.

For right-to-left paragraphs[69], the distance is measured from RightMargin[149].

Collections of tab stops (TRVStyle[385].ParaStyles[398][i].Tabs[455]) are sorted by this property in ascending order. They are resorted automatically when you assign value to this property.

Do not rely on index of item in this collection after assigning this property:

Wrong:

```
ParaStyle.Tabs[455][i].Position := 10;
ParaStyle.Tabs[455][i].Align[470] := rvtaRight; // index of this tab stop may be changed!
```

Correct:

```
ParaStyle.Tabs[455][i].Align[470] := rvtaRight;
ParaStyle.Tabs[455][i].Position := 10;
```

or

```
with ParaStyles.Tabs[455][i] do
begin
  Position := 10;
  Align[470] := rvtaRight;
end;
```

Default value:

0

1.4.2.1.7 Classes of Properties of List Style

1.4.2.1.7.1 TRVListLevelCollection

Collection of list levels. Each level (TRVListLevel[472]) defines properties for bullet or numbering of paragraphs.

This is a class of Levels[459] property of list styles[459].

Unit RVStyle.

Syntax

```
TRVListLevelCollection = class(TCollection)
```

Hierarchy

TObject
TPersistent
TCollection

Main Properties

Items[472] – array of TRVListLevel[472].

Provides indexed access to the items in the collection of paragraph list levels (bullets and numbering)

```
property Items[Index:Integer] : TRVListLevel[472];
```

Each level defines properties for bullet or numbering of paragraphs.

Adds a new style to the collection of list levels.

```
function Add: TRVListLevel[472];
```

This method overrides TCollection.Add method to return TRVListLevel instance.

Call Add to create a new level in the collection. The new item is placed at the end of the Items[472] array.

Add returns the new collection item.

1.4.2.1.7.2 TRVListLevel

One level of paragraph list, defines properties for bullet or numbering. This is a class of item in the Levels[459] collection of list style[459].

Unit RVStyle.

Syntax

```
TRVListLevel = class(TCollectionItem)
```

Hierarchy

TObject
TPersistent
TCollectionItem

Properties

Layout:

- FirstIndent[473], LeftIndent[476] override the corresponding properties of paragraph style;
- MarkerIndent[478] – indent of marker;
- MarkerAlignment[477] – position of marker relative to MarkerIndent.

List marker:

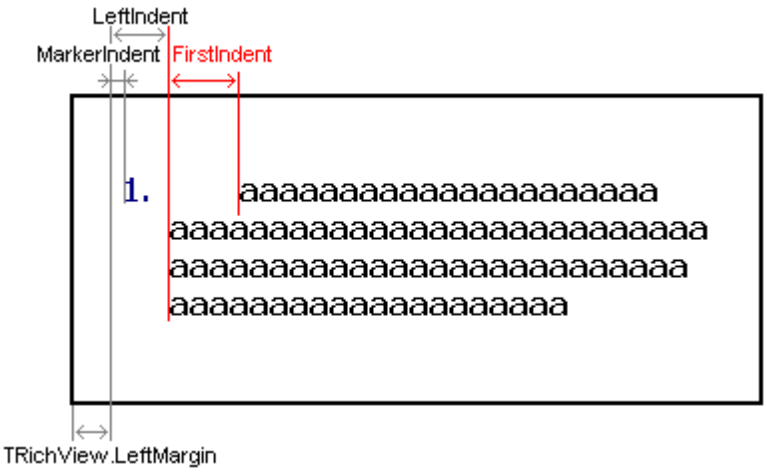
- ListType[476] – type of marker (bullet, number, image, etc.);
- Font[473] – font for text markers;
- Picture[479] – picture, used if ListType[476] =rvlstPicture;
- ImageList[475], ImageIndex[475] – used if ListType[476] in [rvlstImageList, rvlstImageListCounter];
- FormatString[474] – format string for displaying text markers (except for ListType[476]=rvlstUnicodeBullet);
- FormatStringW[475] – text for Unicode bullets (ListType[476]=rvlstUnicodeBullet).

Numbering:

- StartFrom[479] – initial counter value for numbered lists;
- Options[478] – numbering options for this list level.

Indent for the first line of paragraph text, relative to LeftIndent [476].

```
property FirstIndent: TRVStyleLength [644];
```



FirstIndent Example

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units [404] of TRVStyle [385] component which owns the list style.

This value can be negative.

This property overrides FirstIndent [451] of paragraph style for bulleted/numbered paragraphs.

If this value is less than width of marker, text is shifted to the right beyond the FirstIndent to avoid overlapping.

Default value:

10

Note: this default value assumes that RVStyle.Units [404]=*rvstuPixels*, otherwise it could be too small.

See also properties:

- LeftIndent [476],
- MarkerIndent [478].

See also properties of TCustomRichView:

- LeftMargin [145].

Font for text markers.

```
type
 TRVMarkerFont = class (TFont);

 property Font: TRVMarkerFont;
```

This font is used for the following list types [476]:

- *rvlstBullet,*
- *rvlstDecimal,*
- *rvlstLowerAlpha,*
- *rvlstUpperAlpha,*
- *rvlstLowerRoman,*
- *rvlstUpperRoman,*
- *rvlstUnicodeBullet.*

Default value:

'Arial', 8

Defines format for text list markers.

```
property FormatString: String;
```

This property is used for the following list types [476]:

- *rvlstBullet,*
- *rvlstDecimal,*
- *rvlstLowerAlpha,*
- *rvlstUpperAlpha,*
- *rvlstLowerRoman,*
- *rvlstUpperRoman.*

For *rvlstBullet* list type, it completely defines a text to display.

For *rvlstDecimal*..*rvlstUpperRoman*, it defines how to display numbers. The actual string = Format(FormatString, [CL0, CL1, ..., CLN])

where

- CL0 - string containing value of counter of the 0-th level marker;
- CL1 - ----------""" ---------- 1st level marker;
- ...
- CLN - string containing value of counter of this level marker.

Example1:

FormatString = '*' // no numbering

```
* aaaaa
* aaaaa
* aaaaa
```

Example2:

For 0-th level: FormatString = '%s.', ListType=*rvlstDecimal*

For 1-th level: FormatString = '%s.%s.', ListType=*rvlstDecimal*

```
1. aaaaa
2. aaaaa
  2.1. aaaaa
  2.2. aaaaa
3. aaaaa
```

The same result will be with '%0:s.' and '%0:s.%1:s.'

Example3:

For 0-th level: FormatString = 'Chapter %s.', ListType=*rvlstDecimal*

For 1-th level: FormatString = '%1:s)', ListType=*rvlstLowerAlpha*

```
Chapter 1. aaaaa
Chapter 2. aaaaa
  a) aaaaa
  b) aaaaa
Chapter 3. aaaaa
```

The same result will be with 'Chapter %0:s.' and '%1:s)'

Default value:

· (#$B7)

See also properties:

- FormatStringW [475].

Text for Unicode text markers

```
property FormatStringW: WideString;
```

If ListType[476]=*rvlstUnicodeBullet*, this text is displayed.

For Delphi/C++Builder 2009 or newer, this property makes no sense, because FormatString[474] is Unicode too.

Default value:

" (empty string)

See also properties:

- FormatString[474].

Image index for bullets that show image from the image list

```
property ImageIndex: Integer;
```

If ListType[476]=*rvlstImageList*, the marker of this list level is the ImageIndex-th image from ImageList[475].

If ListType[476]=*rvlstImageListCounter*, the marker of this list level is the (Counter+ImageIndex-1)-th image from ImageList[475], there Counter is a value of counter for the given list marker.

List counter is numbered from StartFrom[479].

For example:

Value of Counter	ImageList Index
1	ImageIndex
2	ImageIndex+1
3	ImageIndex+2
4	ImageIndex+3

Default value:

0

See also properties:

- StartFrom[479].

Image list for image of list marker

```
property ImageList: TCustomImageList;
```

ImageList is used If ListType[476] is *rvlstImageList* or *rvlstImageListCounter*.

When saving lists of this type in RVF[59], RichView stores value of ImageList.Tag. When loading, OnRVFImageListNeeded[252] occurs.

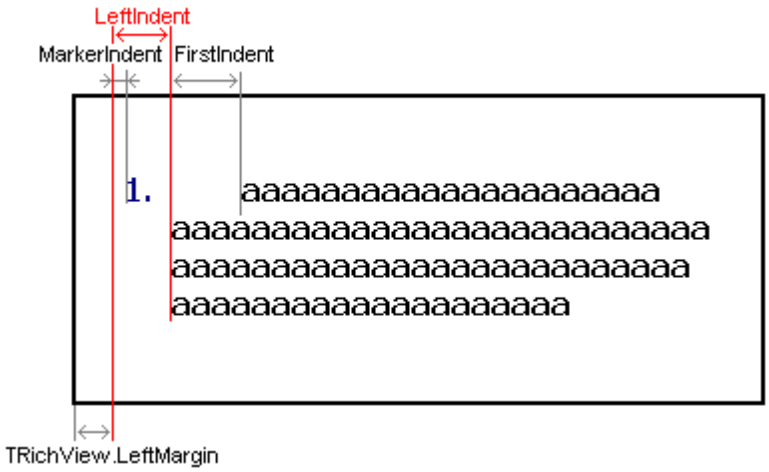
List level does not own this imagelist: it does not create and does not destroy it. It just has a link to it.

See also properties:

- ImageIndex[475];
- Picture[479].

Left indent of paragraph text.

```
property LeftIndent: TRVStyleLength[644];
```



LeftIndent Example

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units[404] of TRVStyle[385] component which owns the list style.

This value can be negative, but it's not recommended.

This property overrides LeftIndent[451] of paragraph style for bulleted/numbered paragraphs. For right-to-left paragraphs[69], list marker is positioned to the right side of paragraph, and this property overrides RightIndent[454] of paragraph style instead.

Default value:

0

See also properties:

- FirstIndent[473],
- MarkerIndent[478].

See also properties of TCustomRichView:

- LeftMargin[145].

Type of list marker for this list level

```
type
  TRVListType = (rvlstBullet, rvlstPicture,
    rvlstImageList, rvlstDecimal,
    rvlstLowerAlpha, rvlstUpperAlpha,
    rvlstLowerRoman, rvlstUpperRoman,
    rvlstImageListCounter,rvlstUnicodeBullet);

property ListType: TRVListType;
```

Unnumbered List Types

List Type	Meaning
rvlstBullet	Unnumbered text (from FormatString[474] property). See also Font[473].
rvlstUnicodeBullet	Unnumbered Unicode text (from FormatStringW[475] property). See also Font[473].

rvlstPicture	Picture (from Picture[479] property)
rvlstImageList	Image from imagelist (ImageList[475] and ImageIndex[475] properties)

Numbered List Types

List Type	Meaning
rvlstDecimal	1, 2, 3,... (see FormatString[474], Font[473])
rvlstLowerAlpha	a, b, c, ..., z, aa, ab, ... (see FormatString[474], Font[473])
rvlstUpperAlpha	A, B, C, ..., Z, AA, AB, ... (see FormatString[474], Font[473])
rvlstLowerRoman	i, ii, iii, iv, ... (see FormatString[474], Font[473])
rvlstUpperRoman	I, II, III, IV, ... (see FormatString[474], Font[473])
rvlstImageListCounter	Image from image list, depending on counter (ImageList[475] and ImageIndex[475] properties)

Default value:

rvlstBullet

Defines marker alignment relative to the MarkerIndent[478]

```
type
  TRVMarkerAlignment = (rvmaLeft, rvmaRight, rvmaCenter);

property MarkerAlignment: TRVMarkerAlignment;
```



MarkerAlignment Example

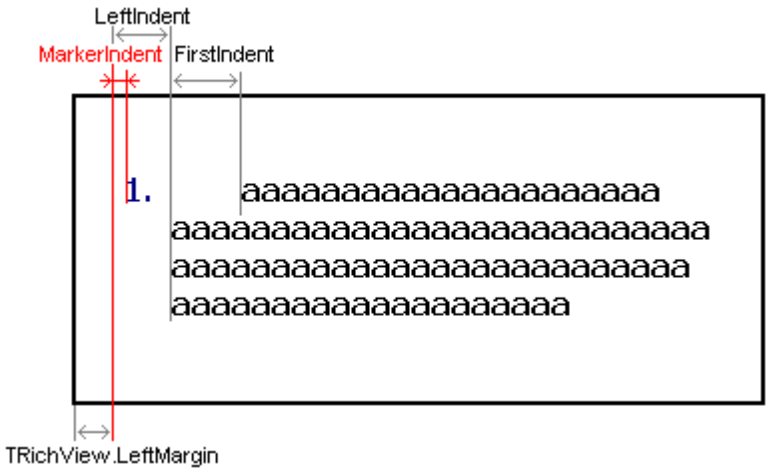
For right-to-left paragraphs[69], list marker position is reversed.

Default value:

rvmaLeft

Indent of the list marker

```
property MarkerIndent: TRVStyleLength[644];
```



MarkerIndent Example

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units[404] of TRVStyle[385] component which owns the list style.

This value can be negative, but it's not recommended.

The image above shows the position of left-aligned marker. See MarkerAlignment[477] property.

Default value:

0

See also properties:

- FirstIndent[473];
- LeftIndent[476].

See also properties of TCustomRichView:

- LeftMargin[145].

List level options

```
type
  TRVListLevelOption =
    (rvloContinuous, rvloLevelReset, rvloLegalStyleNumbering);
  TRVListLevelOptions = set of TRVListLevelOption;
```

If *rvloLevelReset* is in Options, numbering will be like this:

```
  1. aaaa
    a) aaaa
    b) aaaa
  2. aaaa
    a) aaaa
    b) aaaa
```

If it is excluded, numbering will be like this:

```
  1. aaaa
    a) aaaa
    b) aaaa
  2. aaaa
    c) aaaa
    d) aaaa
```

If *rvloLegalStyleNumbering* is in Options, all roman and alpha numbering of higher levels will be changed to decimals, like this:

```
I. aaaa
  1.1 aaaa
  1.2 aaaa
II. aaa
```

Otherwise:

```
I. aaaa
  I.1 aaaa
  I.2 aaaa
II. aaa
```

rvloContinuous is reserved for future use.

Default value:

[*rvloContinuous*, *rvloLevelReset*]

Picture for list marker

```
property Picture: TPicture;
```

This property is used if ListType[476]=*rvlstPicture*.

Starting value for marker counter

```
property StartFrom: Integer;
```

For example, if StartFrom for Levels[459][1]=0:

```
1. aaaaa
  0) aaaaa
  1) aaaaa
2. aaaaa
```

Default value:

1

See also:

- ListType[476].

Returns *True* if this list level is numbered

```
function HasNumbering: Boolean;
```

This method returns

Result := ListType[476] in [*rvlstDecimal, rvlstLowerAlpha, rvlstUpperAlpha*, *rvlstLowerRoman, rvlstUpperRoman, rvlstImageListCounter*].

1.4.2.2 TRVPrint

This component allows printing TRichView[128], TRichViewEdit[282], TDBRichView[373], or TDBRichViewEdit[377]. It is Invisible at run-time.

Unit PtblRV;

Syntax

```
TRVPrint = class(TCustomRVPrint 514)
```

Hierarchy

TObject
TPersistent
TComponent
TCustomRVPrint 514

How to Use

1. Assign the source component for printing (AssignSource 488 method).
2. (optionally) Assign header and/or footer (SetHeader 493, SetFooter 493).
3. Format pages (FormatPages 490 method).
4. Now you can:
   - get a number of pages (PagesCount 516 property);
   - display a print preview (using TRVPrintPreview 384 component; see also DrawPreview 489, MakePreview 491 MakeScaledPreview 491 methods);
   - print the whole document (Print 492 method) or range of pages (PrintPages 492 method).
5. When finished, you can release some temporally allocated memory (Clear 517).

Events

You can display some information about the process of repaginating and printing:

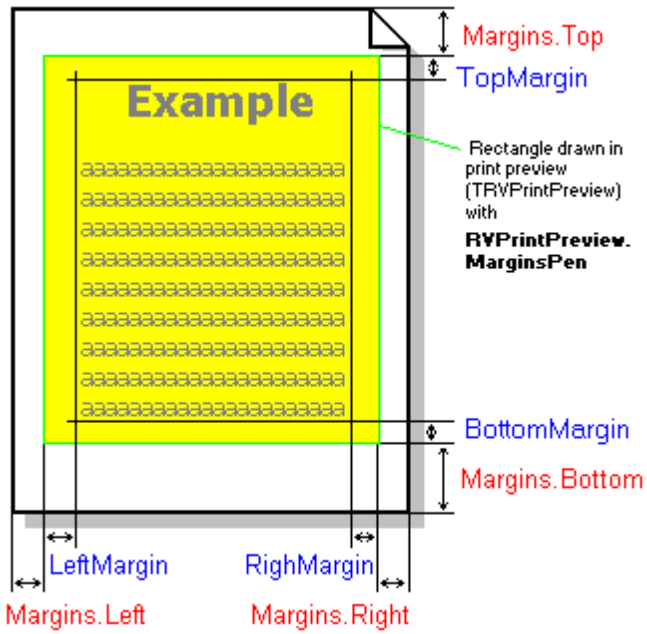
- OnFormatting 494 occurs during repaginating (FormatPages 490 method);
- OnSendingToPrinter 496 occurs during printing (Print 492 and PrintPages 492 methods).

You can draw additional content using the following events:

- OnPagePrepaint 495 and OnPagePostpaint 495 allow to draw on each page below/over TRichView document;
- OnPrintComponent 519 requests image of inserted control for printing.

Margins

Margins are specified in Margins 486 property.



Margins in TRVPrint

This picture illustrates the meaning of margin properties. Paragraph indents are not shown for simplification.

Margins 486 are measured in Units 487 (shown in red).

TRichView.LeftMargin 145, TRichView.RightMargin 149, TRichView.TopMargin 158 and TRichView.BottomMargin 137 are measured in "screen pixels" (they are recalculated according to the printer resolution, and margins on the page look like margins on the screen; shown in blue).

Left and right margins can be swapped for even pages, if MirrorMargins 487 is set to *True*.

See also ClipMargins 484 property.

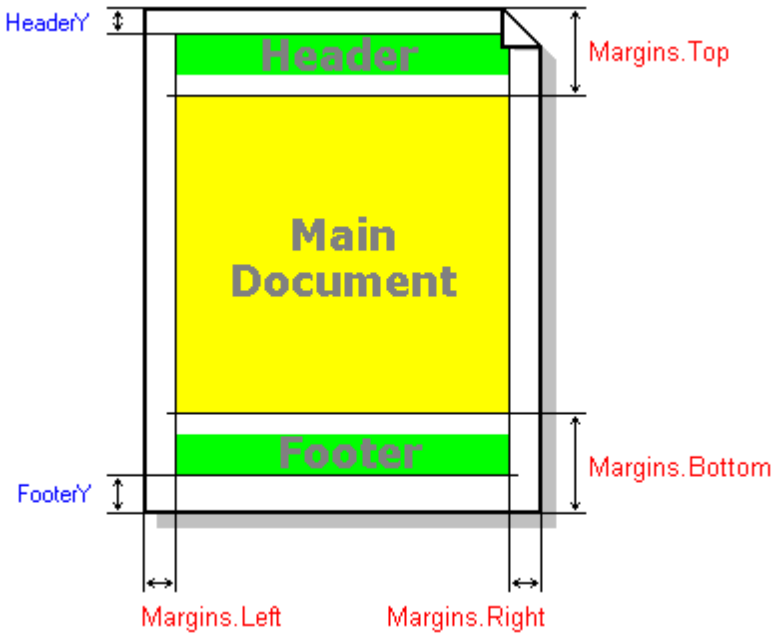
Headers and Footers

TRVPrint supports 3 types of headers and footers:

- normal header and footer
- header and footer for the first page (used only if TitlePage 487=*True*)
- header and footer for even pages (used only if FacingPages 484=*True*)

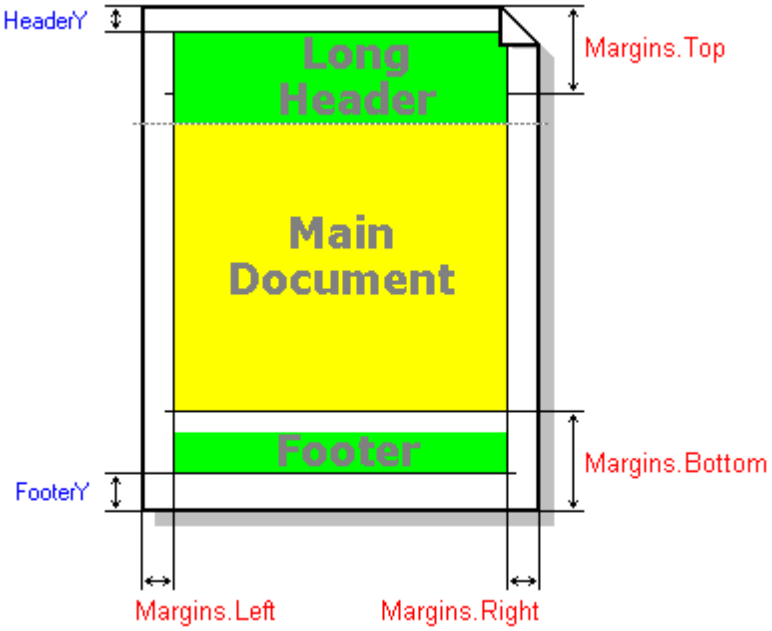
Headers are assigned using SetHeader 493, footers are assigned using SetFooter 493 methods.

Any TRichView document can be assigned as a header or footer. Positions of headers and footers are defined by HeaderY 485 and FooterY 485.



Header and Footer

If a header and/or a footer is too long, size of the main document is adjusted to avoid overlapping:



Long Header Example

If you want more flexible headers / footers, consider:

- printing document in several parts, each of them with its own header/footer;
- using OnPagePrepaint 495 event;
- using TRVReportHelper 503.

See also: GetHeaderRect 490, GetFooterRect 490.

Document inside RVPrint

RVPrint contains an instance of TPrintableRV (descendant of TRichView) inside. This variable (named rv) is public and you can change some properties of it:

- Color 139,
- BackgroundStyle 136,
- BackgroundBitmap 136.

Changing other its properties is not recommended.

Example:

```
MyRVPrint.rv.Color := clWhite;
```

Note: this richview does not contain the full information about the printed document, just a formatting information; so you must provide that source RichView is not destroyed while you are printing.

Incremental printing

You can define the range of items to print, using MinPrintedItemNo 516 and MaxPrintedItemNo 515 properties. If these properties define a subrange of items, only this subrange (and only pages containing it) is printed.

These properties allow implementing an incremental printing: when the next portion of the document is ready, it can be printed below the previously printed fragment. This feature can be used for printing accounting journals, logs, etc.

Order of drawing

The component draws a page content in the following order:

1. page background
2. OnPagePrepaint 495 event
3. Header's text boxes 107 and sidenotes 105 having PositionInText 599=*rvpitBelowText*
4. Header 493
5. Header's text boxes and sidenotes having PositionInText=*rvpitAboveText*
6. Footer's text boxes and sidenotes having PositionInText=*rvpitBelowText*
7. Footer 493
8. Footer's text boxes and sidenotes having PositionInText=*rvpitAboveText*
9. Main document's text boxes and sidenotes having PositionInText=*rvpitBelowText*
10. Main document, including footnotes and endnotes
11. Main document's text boxes and sidenotes having PositionInText=*rvpitAboveText*
12. OnPagePostPaint 495 event

Text boxes and sidenotes are drawn in the order of their positions in the parent document.

Tips and Tricks

See ***http://www.trichview.com/forums/viewtopic.php?p=250***.

See Also

See also:

- UpdatePaletteInfo 518 method;
- TRichView.PageBreaksBeforeItems 149;
- TRichView.AssignSoftPageBreaks 174;
- TRVReportHelper 503 components;
- RichViewAlternativePicPrint 652 typed constant;
- RichViewPixelsPerInch 655 typed constant.

Demo projects:

- Demos\Delphi\Assorted\Printing\
- Demos\CBuilder\Assorted\Printing\

1.4.2.2.1 Properties

1.4.2.2.1.1 TRVPrint.ClipMargins

Disallows printing on margins.

```
property ClipMargins: Boolean
```

(introduced in version 1.7)

If *True*, all drawing that does not fit the printable area (for example, large pictures) is cropped.

Default value:

False

1.4.2.2.1.2 TRVPrint.FacingPages

Enables using different headers and footers on odd and even pages.

```
property FacingPages: Boolean;
```

(introduced in version 15)

If *False*, normal headers and footers are used on all pages.

If *True*, normal headers and footers are used on odd pages, special headers and footers are used on even pages.

If some header or footer is not defined (*nil*), it is not shown.

Default value:

False

See also properties:

- TitlePage [487];
- MirrorMargins [487].

See also methods:

- SetHeader [493];
- SetFooter [493];
- AssignDocParameters [488].

1.4.2.2.1.3 TRVPrint.FixMarginsMode

Specifies how to work with too small Margins [486] (less than the printer supports).

```
type
  TRVFixMarginsMode = (rvfmmAutoCorrect, rvfmmIgnore);
property FixMarginsMode: TRVFixMarginsMode;
```

(introduced in version 10)

Mode	Meaning
rvfmmAutoCorrect	Minimal supported margins are used (if the left and/or the top margin were too small, the output is moved to the right and/or down)
rvfmmIgnore	The specified values of margins are used (document may be cropped)

Default value:

rvfmmAutoCorrect

1.4.2.2.1.4 TRVPrint.FooterY

Vertical position of a page footer.

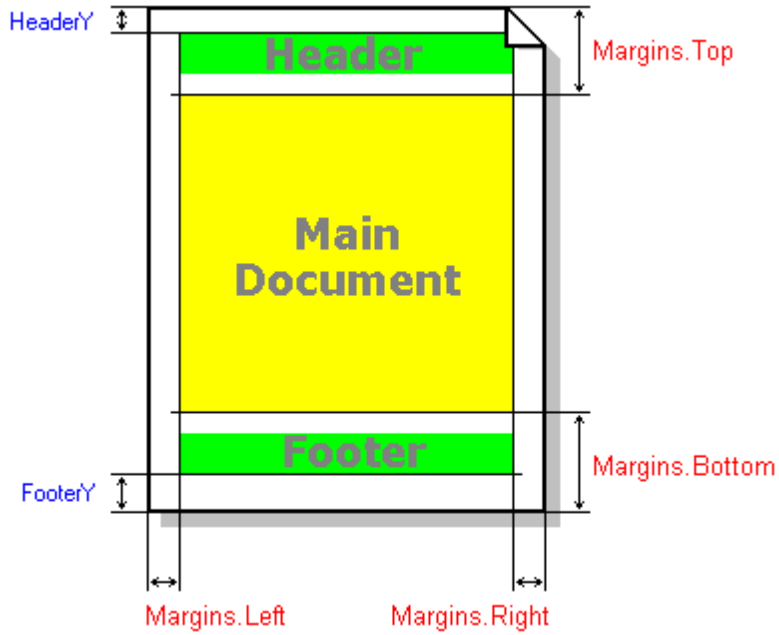
```
property FooterY: TRVLength[636];
```

(introduced in version 15)

This is a distance from the bottom of a page to the bottom of a footer. This value is measured in Units[487].

If a footer is not assigned, this property is ignored.

This property is used in all types of headers (normal, first page, even pages).



Header and Footer

Default value:

10.0

See also:

- HeaderY[485].

See also methods:

- SetFooter[493];
- AssignDocParameters[488].

1.4.2.2.1.5 TRVPrint.HeaderY

Vertical position of a page header.

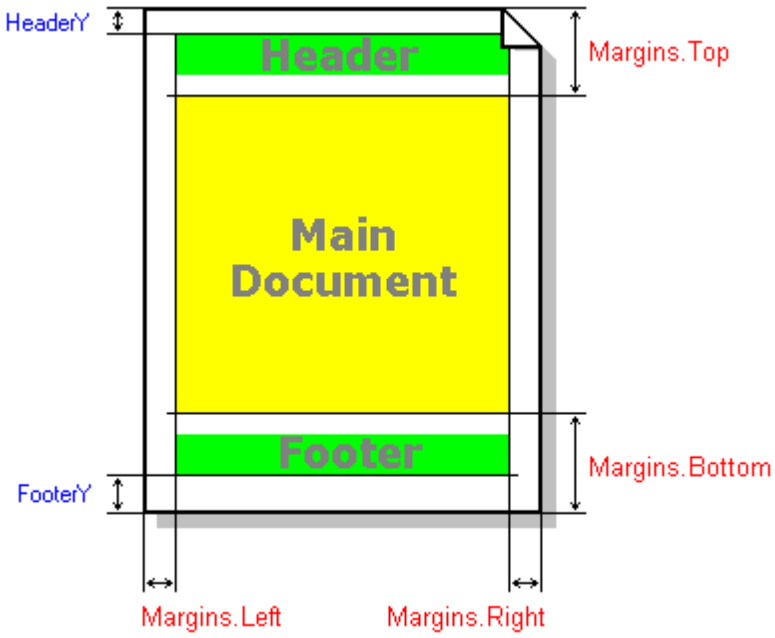
```
property HeaderY: TRVLength[636];
```

(introduced in version 15)

This is a distance from the top of a page to the top of a header. This value is measured in Units[487].

If a header is not assigned, this property is ignored.

This property is used in all types of headers (normal, first page, even pages).



Header and Footer

Default value:

10.0

See also:

- FooterY [485].

See also methods:

- SetHeader [493];
- AssignDocParameters [488].

1.4.2.2.1.6 TRVPrint.Margins

Page margins.

```
property Margins: TRVUnitsRect [615];
```

(introduced in version 15)

Margins are measured in Units [404].

If MirrorMargins [487]=*True*, Margins.Left and Margins.Right are swapped for even pages.

You need to reformat pages after changing values of this property, see FormatPages [490].

See also the picture in the TRVPrint [479] main topic.

Default value:

(20.0, 20.0, 20.0, 20.0)

See also properties:

- HeaderY [485];
- FooterY [485];

See also methods:

- AssignDocParameters [488].

1.4.2.2.1.7 TRVPrint.MirrorMargins

If set to *True*, Margins[486].Left and Margins[486].Right are swapped for even pages.

```
property MirrorMargins: Boolean;
```

(introduced in version 1.6)

Default value:

False

See also properties:

- FacingPages[484].

See also methods:

- AssignDocParameters[488].

1.4.2.2.1.8 TRVPrint.Preview100PercentHeight

Height of a page when drawing a preview on the screen at 100% scale, pixels.

```
property Preview100PercentHeight: Integer;
```

Mainly for internal use.

1.4.2.2.1.9 TRVPrint.Preview100PercentWidth

Width of a page when drawing a preview on the screen at 100% scale, pixels.

```
property Preview100PercentWidth: Integer;
```

Mainly for internal use.

1.4.2.2.1.10 TRVPrint.TitlePage

Enables using different headers on the first page.

```
property FacingPages: Boolean;
```

(introduced in version 15)

If *False*, normal headers and footers are used on the first page.

If *True*, special headers and footers are used on the first pages.

If some header or footer is not defined (*nil*), it is not shown.

Default value:

False

See also properties:

- FacingPages[484].

See also methods:

- SetHeader[493];
- SetFooter[493];
- AssignDocParameters[488].

1.4.2.2.1.11 TRVPrint.Units

Defines measurement units for sizes in TRVPrint.

```
property Units: TRVUnits[648];
```

(introduced in version 15)

The following properties are measured in Units:

- HeaderY[485];

- FooterY 485;
- Margins 486.

A direct assignment to this property does not change values of these properties. To convert properties, use ConvertToUnits 489 method.

Default value:

rvuMillimeters

1.4.2.2.2 Methods

1.4.2.2.2.1 TRVPrint.AssignDocParameters

Assigns properties from **DocParameters**.

```
procedure AssignDocParameters(DocParameters: TRVDocParameters 277);
```

(introduced in version 10)

This procedure assigns the following properties:

- Margins 486;
- HeaderY 485, FooterY 485;
- MirrorMargins 487;
- FacingPages 484;
- TitlePage 487.

Sizes are converted from **DocParameters**.Units 280 to Units 487 (the value of Units 487 property itself is not changed).

This method does not assign printer properties (page size and orientation).

1.4.2.2.2.2 TRVPrint.AssignSource

Assigns document for printing.

```
procedure AssignSource(PrintMe: TCustomRichView 128);
```

You can print document in the following components:

- TRichView 128,
- TRichViewEdit 282,
- TDBRichView 373,
- TDBRichViewEdit 377.

Document is not copied to RVPrint, so do not modify and do not destroy it until printing is completed.

See also:

- SetHeader 493;
- SetFooter 493.

1.4.2.2.2.3 TRVPrint.ContinuousPrint

Prints the whole document in the current printing job.

```
procedure ContinuousPrint;
```

This method is useful if you need to print several documents in one printing job. Otherwise, use Print 492 or PrintPages 492 methods.

Example 1: How to print several documents in one printing job

In this example, the same RVPrint is used to print RichView1 and RichView2. This code starts printing RichView2 on the page where RichView1 ends.

```
Printer.BeginDoc;
// printing RichView1
RVPrint.StartAt 517 := 0;
```

```
RVPrint.TransparentBackground[517] := True;
RVPrint.AssignSource[488](RichView1);
RVPrint.FormatPages[490](rvdoALL);
RVPrint.ContinuousPrint;
// printing RichView2
RVPrint.StartAt[517] := RVPrint.EndAt[514];
RVPrint.AssignSource[488](RichView2);
RVPrint.FormatPages[490](rvdoALL);
RVPrint.ContinuousPrint;
Printer.EndDoc;
```

Example 2: How to mix portrait and landscape orientations in one printing job

In this example, the same RVPrint is used to print RichView1 and RichView2 in different orientations.

```
Printer.BeginDoc;
// printing RichView1 in portrait orientation
Printer.Orientation := poPortrait;
RVPrint.AssignSource[488](RichView1);
RVPrint.FormatPages[490](rvdoALL);
RVPrint.ContinuousPrint;
// printing RichView2 in landscape orientation
Printer.NewPage;
Printer.Orientation := poLandscape;
RVPrint.AssignSource[488](RichView2);
RVPrint.FormatPages[490](rvdoALL);
RVPrint.ContinuousPrint;
Printer.EndDoc;
```

1.4.2.2.2.4 TRVPrint.ConvertToUnits

Converts properties to new units.

```
procedure ConvertToUnits(AUnits: TRVUnits[648]);
```

(introduced in version 15)

The method converts HeaderY[485], FooterY[485], Margins[486] from Units[487] to **AUnits**, then assigns **AUnits** to Units[487].

1.4.2.2.2.5 TRVPrint.Create

A constructor, creates and initializes a TRVPrint instance.

```
constructor Create(AOwner: TComponent);override;
```

Use Create to instantiate TRVPrint object at runtime. TRVPrint objects placed on forms at design time are created automatically. Specify the owner of the new TRVPrint object using the **AOwner** parameter.

1.4.2.2.2.6 TRVPrint.Draw Preview

Draws print preview of the **pgNo**-th page to **PageRect** rectangle of **Canvas**.

```
procedure DrawPreview(pgNo: Integer; Canvas:  TCanvas; const PageRect: TRect);
```

There is more convenient way to display print preview: TRVPrintPreview[384] component.

The page image is scaled to fit **PageRect**. To maintain correct proportions, you can use Preview100PercentWidth[487] and Preview100PercentHeight[487] properties.

pgNo must be in range from 1 to PagesCount[516].

You need to assign source document for printing (AssignSource[488] method) and format pages (FormatPages[490] method) before calling this method.

See also methods:
- MakePreview[491];
- MakeScaledPreview[491].

See also properties:
- PreviewCorrection[516];
- PagesCount[516].

See also events:
- OnPrintComponent[519];
- OnPagePrepaint[495];
- OnPagePostpaint[495].

1.4.2.2.7 TRVPrint.FormatPages

Prepare document to formatting

```
function FormatPages(PrintOptions:TRVDisplayOptions[626]): Integer;
```

Please use *rvdoAll* constant for **PrintOptions** parameter:

```
FormatPages(rvdoAll);
```

Document must be assigned to this TRVPrint object using AssignSource[488] method before calling this method.

This method allocates some temporal memory what is used when drawing preview and printing. When you finish, call Clear[517] to release this memory.

Note for C++Builder users:

C++Builder (at least version 5) incorrectly translates rvdoAll constant.

The recommended call is

```
RVPrint->FormatPages(TRVDisplayOptions())
```

(i.e. initialization with the empty set; this parameter is for compatibility, and it is not used any more)

See also methods:
- Clear[517].

See also events:
- OnFormatting[494].

1.4.2.2.8 TRVPrint.GetFooterRect

Returns coordinates of the footer for the page **PageNo** (from 1), in printer pixels

```
function GetFooterRect(PageNo: Integer): TRect;
```

(introduced in version 1.7; the parameter is added in version 15)

If the footer is empty, the function returns an empty rectangle (Rect(0,0,0,0)).

This function may be called only when the document is prepared for printing (see FormatPages[490]).

This function can be used in OnPagePrepaint[495] and OnPagePostpaint[495] events.

See also:
- SetFooter[493];
- GetHeaderRect[490].

1.4.2.2.9 TRVPrint.GetHeaderRect

Returns coordinates of the header for the page **PageNo** (from 1), in printer pixels

```
function GetHeaderRect(PageNo: Integer): TRect;
```

(introduced in version 1.7; the parameter is added in version 15)

If the header is empty, the function returns an empty rectangle (Rect(0,0,0,0)).

This function may be called only when the document is prepared for printing (see FormatPages 490)

This function can be used in OnPagePrepaint 495 and OnPagePostpaint 495 events.

See also:

- SetHeader 493;
- GetFooterRect 490.

1.4.2.2.2.10 TRVPrint.MakePreview

Draws a preview of the **pgNo**-th page to bitmap **bmp**.

```
procedure MakePreview(pgNo: Integer; bmp: TBitmap);
```

There is more convenient way to display print preview: TRVPrintPreview 384 component.

pgNo must be in range from 1 to PagesCount 516.

This method changes size of bitmap to (Preview100PercentWidth 487 x Preview100PercentHeight 487) and draws print preview of the **pgNo**-th page into it.

You need to assign source document for printing (AssignSource 488 method) and format pages (FormatPages 490 method) before calling this method.

See also methods:

- MakeScaledPreview 491;
- DrawPreview 489.

See also properties:

- PreviewCorrection 516;
- PagesCount 516.

See also events:

- OnPrintComponent 519;
- OnPagePrepaint 495;
- OnPagePostpaint 495.

1.4.2.2.2.11 TRVPrint.MakeScaledPreview

Draws print preview of the **phNo**-th page to bitmap **bmp**.

```
procedure MakeScaledPreview(pgNo: Integer; bmp: TBitmap);
```

There is more convenient way to display print preview: TRVPrintPreview 384 component.

pgNo must be in range from 1 to PagesCount 516.

This method does not change size of bitmap, and draws the page in full bitmap size. To maintain correct proportions, you can use Preview100PercentWidth 487 and Preview100PercentHeight 487 properties.

You need to assign source document for printing (AssignSource 488 method) and format pages (FormatPages 490 method) before calling this method.

See also methods:

- MakePreview 491;
- DrawPreview 489.

See also properties:

- PreviewCorrection 516;
- PagesCount 516.

See also events:

- OnPrintComponent 519;
- OnPagePrepaint 495;

- OnPagePostpaint[495].

1.4.2.2.2.12 TRVPrint.Print

Prints the whole document

```
procedure Print(Title: String; Copies: Integer; Collate: Boolean);
```

This method takes MinPrintedItemNo[516] and MaxPrintedItemNo[515] properties into account.

Parameters

Title – the text that is listed in the Print Manager when printing.

Copies – specifies the number of copies to print.

Collate – if more than one copy is selected, specifies whether you want the copies to be collated.

Assign the source document for printing (AssignSource[488] method) and format pages (FormatPages[490] method) before calling this method.

```
Print(Title, Copies, Collate)
```

is equivalent to

```
PrintPages[492](1, PagesCount[516], Title, Copies, Collate)
```

See also methods:

- PrintPages[492];

See also events:

- OnSendingToPrinter[496];
- OnPrintComponent[519];
- OnPagePrepaint[495];
- OnPagePostpaint[495].

1.4.2.2.2.13 TRVPrint.PrintPages

Prints the range of pages, from **FirstPgNo** to **LastPgNo**

```
procedure PrintPages(FirstPgNo, LastPgNo: Integer; Title: String;
  Copies: Integer; Collate: Boolean);
```

This method takes MinPrintedItemNo[516] and MaxPrintedItemNo[515] properties into account.

Parameters

FirstPgNo – the first page to print, must be in range 1..PagesCount[516].

LastPgNo – the last page to print, must be in range 1..PagesCount[516].

Title – the text that is listed in the Print Manager when printing.

Copies – specifies the number of copies to print.

Collate – if more than one copy is selected, specifies whether you want the copies to be collated.

You need to assign source document for printing (AssignSource[488] method) and format pages (FormatPages[490] method) before calling this method.

See also methods:

- Print[492].

See also events:

- OnSendingToPrinter[496];
- OnPrintComponent[519];
- OnPagePrepaint[495];
- OnPagePostpaint[495].

1.4.2.2.2.14 TRVPrint.SetFooter

Assigns a page footer.

```
procedure SetFooter(RVData: TCustomRVData[608]; FooterType: TRVHFType[634]=rvhftNormal);
```

(introduced in version 1.7; modified in version 15)

Parameters:

RVData – document to display as a footer of the type specified in **HFType**; the document is not copied to RVPrint, so do not modify and do not destroy it until printing is completed. If **RVData** is *nil*, a footer of this type is not displayed.

HFType – footer type (normal/for the first page/for even pages).

This method must be called before FormatPages[490].

Using different footers

A footer for the first page is used only if TitlePage[487]=*True*, otherwise it is ignored, and a normal footer is displayed on the first page.

A special footer for even pages is used only if FacingPages[484]=*True*, otherwise it is ignored, and a normal footer is used on both odd and even pages.

Example:

```
var
  rvMain, rvFooter: TRichView[128];
  RVPrint1: TRVPrint;
...
RVPrint1.AssignSource(rvMain);
RVPrint1.SetFooter(rvFooter.RVData[151]);
RVPrint1.TitlePage[487] := True;
RVPrint1.SetFooter(nil, rvhftFirstPage); // hiding footer on the first page
```

See also:

- SetHeader[493];
- AssignSource[488].

See also properties:

- FooterY[485].

See also:

- TRichView[128].RTFReadProperties[151].SetFooter[276].

1.4.2.2.2.15 TRVPrint.SetHeader

Assigns a page header.

```
procedure SetHeader(RVData: TCustomRVData[608]; FooterType: TRVHFType[634]=rvhftNormal);
```

(introduced in version 1.7; modified in version 15)

Parameters:

RVData – document to display as a header of the type specified in **HFType**; the document is not copied to RVPrint, so do not modify and do not destroy it until printing is completed. If **RVData** is *nil*, a header of this type is not displayed.

HFType – header type (normal/for the first page/for even pages).

This method must be called before FormatPages[490].

Using different headers

A header for the first page is used only if TitlePage[487]=*True*, otherwise it is ignored, and a normal header is displayed on the first page.

A special header for even pages is used only if FacingPages[484]=*True*, otherwise it is ignored, and a normal header is used on both odd and even pages.

Example:

```
var
  rvMain, rvHeader: TRichView[128];
  RVPrint1: TRVPrint;
...
RVPrint1.AssignSource(rvMain);
RVPrint1.SetHeader(rvHeader.RVData[151]);
RVPrint1.TitlePage[487] := True;
RVPrint1.SetHeader(nil, rvhftFirstPage); // hiding header on the first page
```

See also:

- SetFooter[493];
- AssignSource[488].

See also properties:

- HeaderY[485].

See also:

- TRichView[128].RTFReadProperties[151].SetHeader[276].

1.4.2.2.3 Events

1.4.2.2.3.1 TRVPrint.OnFormatting

Occurs while FormatPages[490] method is executed.

```
type
  TRVPrintingEvent = procedure(Sender: TCustomRichView[128];
    PageCompleted: Integer;
    Step:TRVPrintingStep[638]) of object;

property OnFormatting: TRVPrintingEvent;
```

In the first time, this event is generated with

PageCompleted=0, **Step**=*rvpsStarting*.

Next, this event occurs several times with

PageCompleted=<number of page which was formatted>, **Step**=*rvpsProceeding*.

In the last time, it occurs with

PageCompleted=0, **Step**=*rvpsFinished.*

Note1: Sender is a special RichView, internally contained in TRVPrint object.

Note2: The first step (preliminary formatting) takes 90% of time; subsequent repagination is quick.

See also methods:

- FormatPages[490].

See also events:

- OnSendingToPrinter[496].

1.4.2.2.3.2 TRVPrint.OnPagePostpaint

Allows to draw on a page (paper and print preview)

```
type
  TRVPagePrepaintEvent = procedure (Sender: TRVPrint;
    PageNo: Integer; Canvas: TCanvas; Preview: Boolean;
    PageRect, PrintAreaRect: TRect) of object;

property OnPagePostpaint: TRVPagePrepaintEvent;
```

(introduced in version 1.7)

Sequence of actions:

1. drawing background (if not transparent);
2. OnPagePrepaint 495;
3. drawing document;
4. OnPagePostpaint.

Parameters

PageNo – page number (starting from 1);

Canvas – canvas where to draw;

Preview – "is this a preview drawing?"

PageRect – rectangle around the full page, pixels.

PrintAreaRect – rectangle around the document on the page (**PageRect** minus marginsMM), pixels.

See also:

- GetHeaderRect 490;
- GetFooterRect 490;
- the picture in the TRVPrint 479 main topic.

1.4.2.2.3.3 TRVPrint.OnPagePrepaint

Allows to draw on a page (paper and print preview)

```
type
  TRVPagePrepaintEvent = procedure (Sender: TRVPrint;
    PageNo: Integer; Canvas: TCanvas; Preview: Boolean;
    PageRect, PrintAreaRect: TRect) of object;

property OnPagePrepaint: TRVPagePrepaintEvent;
```

(introduced in version 1.3)

Sequence of actions:

1. drawing background (if not transparent);
2. OnPagePrepaint;
3. drawing document;
4. OnPagePostpaint 495.

Parameters

PageNo – page number (starting from 1);

Canvas – canvas where to draw;

Preview – "is this a preview drawing?"

PageRect – rectangle around the full page, pixels.

PrintAreaRect – rectangle around the document on the page (**PageRect** minus marginsMM), pixels.

See also:

- GetHeaderRect 490;
- GetFooterRect 490;
- the picture in the TRVPrint 479 main topic;
- example how to print pictures in this event: ***http://www.trichview.com/forums/viewtopic.php?t=58***.

1.4.2.3.4 TRVPrint.OnSendingToPrinter

Occurs while methods Print 492 or PrintPages 492 are executed.

```
type
  TRVPrintingEvent = procedure(Sender: TCustomRichView 128;
    PageCompleted: Integer; Step:TRVPrintingStep 638) of object;

property OnSendingToPrinter: TRVPrintingEvent;
```

In the first time, this event is generated with

PageCompleted=0, **Step**=*rvpsStarting*.

Next, this event occurs several times with

PageCompleted=<number of page that was just printed>, **Step**=*rvpsProceeding*.

In the last time, it occurs with

PageCompleted=0, **Step**=*rvpsFinished.*

Note1: Sender is a special RichView, internally contained in TRVPrint object.

Note2: PageCompleted is an index of page (from 1), not a number of printed pages. The same value will be repeated several times in one call of Print or PrintPages when printing more than one copy.

See also methods:

- Print 492;
- PrintPages 492.

See also events:

- OnFormatting 494.

1.4.2.3 TRVOfficeConverter

TRVOfficeConverter allows using text converters from Microsoft Office (32-bit versions) with RichViews, for importing and exporting files in different formats.

Unit RVOfficeCnv;

Syntax

```
TRVOfficeConverter = class (TComponent)
```

(introduced in version 1.6)

Hierarchy

TObject
TPersistent
TComponent

How to Use

There are two types of converters: for importing and for exporting files.

Export converters convert RTF (Rich Text Format) to other formats, import converters convert different formats to RTF.

Lists of available converters are in ImportConverters 499 and ExportConverters 499 properties.

If there are no converters installed, converters lists are empty.

Note: some converters are shipped with Windows.

GetImportFilter 501 and GetExportFilter 501 return strings for assigning to Filter property of file-selection dialogs.

There are two modes for using this components.

1. You can convert RTF from Stream 500 to file in another format (ExportRTF 500), or you can convert file to RTF Stream (ImportRTF 501).
2. You can export content of TRichView component in a file (ExportRV 500) or import from file (ImportRV 502). Such conversion still uses Stream 500 internally.

While converting, the component generates OnConverting 503 event.

If PreviewMode 499 property is set to *True*, converters do their work silently, without displaying any dialogs. They can also produce lower quality results in this mode (that depends on a converter).

Platform notes

This component is useful only in 32-bit applications, because converters are 32-bit DLLs.

Demo projects

- Demos\Delphi\OfficeConverters\
- Demos\CBuilder\OfficeConverters\

Additional resources

- ***Microsoft Office Compatibility Pack for Word, Excel, and PowerPoint File Formats*** contains a set of converters for new formats of Microsoft Office

1.4.2.3.1 Properties

1.4.2.3.1.1 TRVOfficeConverter.ErrorCode

Returns error code of the last performed import or export operation (if it failed)

```
property ErrorCode: Integer;
```

(introduced in version 1.7)

Error codes returned by the component:

Constant	Value	Meaning
rvceCnvLoadError	1	error loading converter's DLL
rvceFuncError	2	required function is not found in the converter's DLL
rvceInitError	3	converter initialization failure

Some (not all) error codes returned by converters:

Constant	Value	Meaning
rvceOpenInFileErr	-1	could not open input file
rvceReadErr	-2	error during read
rvceOpenConvErr	-3	error opening conversion file
rvceWriteErr	-4	error during write
rvceInvalidFile	-5	invalid data in conversion file
rvceOpenExceptErr	-6	error opening exception file
rvceWriteExceptErr	-7	error writing exception file
rvceNoMemory	-8	out of memory
rvceInvalidDoc	-9	invalid document
rvceDiskFull	-10	out of space on output
rvceDocTooLarge	-11	conversion document too large for target
rvceOpenOutFileErr	-12	could not open output file
rvceUserCancel	-13	conversion canceled by user
rvceWrongFileType	-14	wrong file type for this converter

1.4.2.3.1.2 TRVOfficeConverter.ExcludeDocX-Converter

These properties allow to exclude DocX converters from the list.

```
property ExcludeDocXImportConverter: Boolean;
property ExcludeDocXExportConverter: Boolean;
```

(introduced in version 15)

If ExcludeDocXImportConverter=*True*, DocX import converter is excluded from the list of ImportConverters [499].
If ExcludeDocXExportConverter=*True*, DocX export converter is excluded from the list of ExportConverters [499] (TRichView has SaveDocX [215] method allowing to save DocX better than this converter).

Default value

False

1.4.2.3.1.3 TRVOfficeConverter.ExcludeHTML-Converter

These properties allow to exclude HTML converters from the list.

```
property ExcludeHTMLImportConverter: Boolean;
property ExcludeHTMLExportConverter: Boolean;
```

(introduced in version 1.7)

If ExcludeHTMLImportConverter=*True*, HTML import converter is excluded from the list of ImportConverters [499] (useful, if you have other function for HTML import).
If ExcludeHTMLExportConverter=*True*, HTML export converter is excluded from the list of ExportConverters [499] (TRichView has methods (SaveHTML [216] and SaveHTMLEx [217]) allowing to save HTML better than this converter).

Default value

False

1.4.2.3.1.4 TRVOfficeConverter.ExportConverters

Collection of installed export converters. Export converters convert RTF to foreign formats.

```
property ExportConverters: TRVOfficeCnvList;
```

This is a list of TRVOfficeConverterInfo.

Order of converters in the list is arbitrary: on another computer, even if the same converters are installed, they may be listed in another order.

See also:

- ImportConverters property;
- GetExportFilter method.

1.4.2.3.1.5 TRVOfficeConverter.ExtensionsInFilter

Allows including file extensions in the file filter (GetImportFilter and GetExportFilter)

```
property ExtensionsInFilter: Boolean;
```

(introduced in version 1.7)

For example:

- if ExtensionsInFilter=*False*, file filter entry would be 'Windows Write';
- if ExtensionsInFilter=*True*, file filter entry would be 'Windows Write (*.wri)';

Default value:

False

1.4.2.3.1.6 TRVOfficeConverter.ImportConverters

Collection of installed import converters. Import converters convert foreign formats to RTF.

```
property ImportConverters: TRVOfficeCnvList;
```

This is a list of TRVOfficeConverterInfo.

Order of converters in the list is arbitrary: on another computer, even if the same converters are installed, they may be listed in another order.

See also:

- ExportConverters property;
- GetImportFilter method.

1.4.2.3.1.7 TRVOfficeConverter.PreviewMode

Instructs converters to do their work without displaying dialogs

```
property PreviewMode: Boolean;
```

(introduced in v1.9)

If set to *True*, converters do their work silently, without displaying any dialogs. They can also produce lower quality results in this mode (that depends on converter).

Default value:

False

1.4.2.3.1.8 TRVOfficeConverter.Stream

A stream to store RTF (Rich Text Format) data.

```
property Stream: TMemoryStream;
```

Export converters convert RTF from this stream to file, import-converters convert file to this RTF stream.

The methods ExportRTF[500] and ImportRTF[501] work with this stream directly.

The methods ExportRV[500] and ImportRV[502] use this stream internally.

1.4.2.3.2 Methods

1.4.2.3.2.1 TRVOfficeConverter.Create

A constructor, creates a new TRVOfficeConverter component.

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

Use Create to instantiate TRVOfficeConverter object at runtime. RVOfficeConverters placed on forms at design time are created automatically. Specify the owner of the new RVOfficeConverter using the **AOwner** parameter.

1.4.2.3.2.2 TRVOfficeConverter.Destroy

Destroys the RVOfficeConverter

```
destructor Destroy;
```

1.4.2.3.2.3 TRVOfficeConverter.ExportRTF

Exports RTF from Stream[500] to file in foreign format.

```
function ExportRTF(const FileName: String; ConverterIndex: Integer): Boolean;
```

Parameters:

FileName is a name of file to save.

ConverterIndex is an index of export converter (in ExportConverters[499] list).

When calling this method, Stream[500] must contain RTF (Rich Text Format) document. The method start conversion from Stream[500].Position and stopped when reached the end of stream.

You can provide a visual indication for the conversion progress using OnConverting[503] event.

Return value:

True if the export was successful, *False* if not.

See also:

- ImportRTF[501];
- ExportRV[500].

See also properties:

- ErrorCode[497].

1.4.2.3.2.4 TRVOfficeConverter.ExportRV

Exports content of **rv** to file in foreign format.

```
function ExportRV(const FileName: String; rv: TCustomRichView;
  ConverterIndex: Integer): Boolean;
```

Parameters:

FileName is a name of file to save.

ConverterIndex is an index of export converter (in ExportConverters[499] list)

This method:

1. clears Stream 500;
2. saves RTF to Stream 500 using **rv**.SaveRTFToStream 220;
3. performs converting;
4. clears Stream 500.

You can provide a visual indication for the conversion progress using OnConverting 503 event.

Return value:

True if the export was successful, *False* if not.

See also:

- ImportRTF 501;
- ExportRV 500.

See also properties:

- ErrorCode 497.

1.4.2.3.2.5 TRVOfficeConverter.GetExportFilter

Returns string that can be assigned to the Filter property of TSaveDialog.

```
function GetExportFilter: String;
```

If ExtensionsInFilter 499 is *True*, file extensions are included in the visible part of the returned file filter.

See also:

- GetImportFilter 501 method;
- ExportConverters 499 property.

1.4.2.3.2.6 TRVOfficeConverter.GetImportFilter

Returns string that can be assigned to the Filter property of TOpenDialog

```
function GetImportFilter: String;
```

If ExtensionsInFilter 499 is *True*, file extensions are included in the visible part of the returned file filter.

See also:

- GetExportFilter 501 method;
- ImportConverters 499 property.

1.4.2.3.2.7 TRVOfficeConverter.ImportRTF

Converts data from the file in foreign format to RTF data in Stream 500

```
function ImportRTF(const FileName: String;
  ConverterIndex: Integer): Boolean;
```

Parameters:

FileName is a name of file to import.

ConverterIndex is an index of import converter (in ImportConverters 499 list)

The method clears Stream 500 and writes RTF (Rich Text Format) document in it.

You can provide a visual indication for the conversion progress using OnConverting 503 event.

Return value:

True if the import was successful, *False* if not.

See also:

- ExportRTF 500;
- ImportRV 502.

See also properties:

- ErrorCode 497.

1.4.2.3.2.8 TRVOfficeConverter.ImportRV

Imports content of file in foreign format to **rv**.

```
function ImportRV(const FileName: String; rv: TCustomRichView;
  ConverterIndex: Integer): Boolean;
```

Parameters:

FileName is a name of file to import.

ConverterIndex is an index of import converter (in ImportConverters 499 list)

This method:

1. clears Stream 500;
2. imports data from the file to Stream 500;
3. reads data from Stream 500 in **rv**, using **rv**.LoadRTFFromStream 210,
4. clears Stream 500.

You can provide a visual indication for the conversion progress using OnConverting 503 event.

Return value:

True if the import was successful, *False* if not.

See also:

- ImportRTF 501;
- ExportRV 500.

See also properties:

- ErrorCode 497.

1.4.2.3.2.9 TRVOfficeConverter.IsValidImporter

Checks if the specified file can be imported with the specified converter.

```
function IsValidImporter(const FileName: String; Index: Integer): Boolean;
```

(introduced in version 12)

Parameters:

FileName is a name of file to test.

Index is an index of import converter (in ImportConverters 499 list)

Return value:

If this method returns *False*, this file cannot be imported by this converter. If this method returns *True*, this file **probably** can be imported by this converter.

See also properties:

- ErrorCode 497.

1.4.2.3.3 Events

1.4.2.3.3.1 TRVOfficeConverter.OnConverting

Occurs while converting (in ImportRTF [501], ExportRTF [500], ImportRV, [502] ExportRV [500] methods)

```
property OnConverting: TConvertingEvent;
type
  TConvertingEvent = procedure (Sender:TObject; Percent: Integer) of object;
```

Percent is a value of completeness, from 0 to 100.

See also event of TRichView:

- OnProgress [248].

1.4.2.3.4 Classes of Properties

1.4.2.3.4.1 TRVOfficeCnvList

This class represents a list of installed import [499] or export [499] converters.

Unit RVOfficeCnv;

Syntax

```
TRVOfficeCnvList = class (TList)
```

Hierarchy

TObject
TList

Main Properties

The main properties are **Count** (count of converters) and **Items** (array of TRVOfficeConverterInfo [503]).

1.4.2.3.4.2 TRVOfficeConverterInfo

This class represents an item in list of installed import [499] and export [499] converters.

Unit RVOfficeCnv;

Syntax

```
TRVOfficeConverterInfo = class;
```

Hierarchy

TObject

Main Properties

This class has the following main properties:

- **Name**: String – name of format (for example, 'HTML Document');
- **Path**: String – path to the converter's DLL;
- **Filter**: String – filter (file mask, for example '*.htm;*.html;*.htx;*.otm').

1.4.2.4 TRVReportHelper

This component is similar to TRVPrint [479], but:

- it can be used to draw on any Canvas (not only on the printer's canvas) pages of any size (not limited to printer page sizes);
- it contains a document, not a reference to it.

Unit RVReport;

Syntax

```
TRVReportHelper = class(TCustomRVPrint[514])
```

Hierarchy

TObject
TPersistent
TComponent
TCustomRVPrint[514]

How to Use

1. Load document in RVReportHelper.RichView[505].
2. Initialize formatting (Init[507]). Specify the page width and canvas.
3. Format the document page by page (FormatNextPage[506]). Pages may have different heights.
4. Draw pages on canvas (DrawPage[505] or DrawPageAt[506]). If you need to know positions of hyperlinks and checkpoints, use OnDrawHyperlink[508] and OnDrawCheckpoint[507] events.

Example (drawing document in metafile)

```
procedure TForm1.DrawToMetafile(rvh: TRVReportHelper;
  wmf: TMetafile; Width: Integer);
const VERYLARGEVALUE = $FFFFFFF;
var Canvas: TMetafileCanvas;
begin
  rvh.IgnorePageBreaks[515] := True;
  rvh.Init[507](Self.Canvas, Width);
  while rvh.FormatNextPage[506](VERYLARGEVALUE) do;
  wmf.Width := Width;
  wmf.Height := rvh.GetLastPageHeight[506];
  Canvas := TMetafileCanvas.Create(wmf, 0);
  rvh.DrawPage[505](1,Canvas,True,rvh.GetLastPageHeight[506]);
  Canvas.Free;
end;
```

Demo projects

- Demos\Delphi\Assorted\Graphics\ToImage\
- Demos\Delphi\Assorted\Printing\ReportHelper\

(for C++Builder demos, change "Delphi" to "CBuilder" in the paths above)

1.4.2.4.1 Properties

1.4.2.4.1.1 TRVReportHelper.AllowTransparentDrawing

Allows using methods drawing semitransparent images

```
property AllowTransparentDrawing: Boolean;
```

(introduced in version 16)

This property is not used when printing, i.e. it is used when **Preview** parameter of DrawPage[505]/DrawPageAt[506] is *True*.

AllowTransparentDrawing	"Preview"	Result
True	*True*	Drawing using semitransparent methods
False	*True*	Drawing semitransparent areas in a bitmap, then drawing this bitmap

True or False	*False*	Drawing semitransparent areas in a bitmap, then printing this bitmap (a special printer-friendly method of bitmap drawing is used)

Default value

False

1.4.2.4.1.2 TRVReportHelper.NoMetafiles

Allows drawing metafiles as bitmaps.

```
property NoMetafiles: Boolean;
```

(introduced in version 15)

If *True*, metafiles (in pictures [90] and *hot-pictures* [93]) are drawn as bitmap images.

It may be useful for PDF creators, if they have problems with metafiles.

Default value

False

1.4.2.4.1.3 TRVReportHelper.RichView

This is the object containing document of the report helper.

```
property RichView: TRichView[128];
```

This RichView object is invisible. Use it to load document and to set properties for loading. This RichView must be linked [156] with some TRVStyle [385] object.

In D6+, you can set properties and events of this object at designtime using the Object Inspector.

Do not use formatting methods of this RichView. Use Init [507] and FormatNextPage [506] instead.

1.4.2.4.2 Methods

1.4.2.4.2.1 TRVReportHelper.DrawPage

Draws the specified page

```
procedure DrawPage(APageNo: Integer; ACanvas: TCanvas;
  APreview: Boolean; AHeight: Integer);
```

Parameters:

APageNo – index of the page to draw. The first page has index = 1.

ACanvas – Canvas where to render the page. It's not necessary the same canvas as was used in Init [507], but must have the same resolution (and the same value of Font.PixelsPerInch)

The top left corner of the page is drawn at (0,0) coordinates.

APreview – should be set to *True* when drawing on the screen canvas, to *False* when drawing on printer canvas.

AHeight – height of this page; this value is used for drawing background.

This page must be formatted before drawing, see FormatNextPage [506].

See also:

- DrawPageAt [506].

1.4.2.4.2.2 TRVReportHelper.DrawPageAt

Draws the specified page at the specified coordinates

```
procedure DrawPageAt(Left, Top, APageNo: Integer;
  ACanvas: TCanvas; APreview: Boolean; AHeight: Integer);
```

Parameters:

APageNo – index of the page to draw. The first page has index = 1.

ACanvas – Canvas where to render the page. It's not necessary the same canvas as was used in Init [507], but must have the same resolution (and the same value of Font.PixelsPerInch)

The top left corner of the page is drawn at (**Left**,**Top**) coordinates.

APreview – should be set to *True* when drawing on the screen canvas, to *False* when drawing on printer canvas.

AHeight – height of this page; this value is used for drawing background.

This method uses SetWindowOrgEx to move origin to (**Left**,**Top**).

This page must be formatted before drawing, see FormatNextPage [506].

See also:

- DrawPage [505].

1.4.2.4.2.3 TRVReportHelper.Finished

Returns *True* if all pages were formatted.

```
function Finished: Boolean;
```

You can use the value returned by FormatNextPage [506] instead.

1.4.2.4.2.4 TRVReportHelper.FormatNextPage

Formats next page(s) of document

```
function FormatNextPage(AMaxHeight: Integer): Boolean;
```

Call this method after Init [507] several times, while it returns *True*.

Parameter:

AMaxHeight – page height, pixels; page width is specified as a parameter of Init [507].

When the page is formatted, it can be drawn (DrawPage [505], DrawPageAt [506]).

Sometimes, this method formats more than one page.

This method is fast. The most time consuming method is Init [507].

Return value:

- *True* if not all pages were formatted;
- *False*, if formatting is complete.

1.4.2.4.2.5 TRVReportHelper.GetLastPageHeight

Returns a real height of the last page, in device pixels

```
function GetLastPageHeight: Integer;
```

This function returns height of document on the last page, when the document is completely formatted (all calls to FormatNextPage [506] are made).

The returned value is measured in **ACanvas** pixels, where **ACanvas** is the parameter of Init 507.

This method is useful if you draw/print several documents one after another.

1.4.2.4.2.6 TRVReportHelper.Init

Prepares document for formatting.

```
procedure Init(ACanvas: TCanvas; APageWidth: Integer);
```

Parameters:

ACanvas – canvas object used for formatting.

APageWidth – page width, pixels; all pages have the same width.

See also:

- FormatNextPage 506.

1.4.2.4.2.7 TRVReportHelper.UpdatePageNumbers

Recalculates page numbers.

```
procedure UpdatePageNumbers;
```

(introduced in version 15)

This method recalculates values of "page number" 109 and "page count" 110 fields.

Normally, values of these fields are calculated when the last page is formatted.

This procedure needs to be called if values of "page number" field are changed. For example, if this RVReportHelper is used to draw the same content on each page, you can provide a number of this page in OnGetPageNumber 508 event, and call UpdatePageNumber before drawing.

1.4.2.4.3 Events

1.4.2.4.3.1 TRVReportHelper.OnDraw Checkpoint

Occurs when the ReportHelper draws item having *checkpoint* 30

```
type
  TRVDrawCheckpointEvent = procedure (Sender: TRVReportHelper;
    RVData: TCustomRVData 608; ItemNo: Integer; X, Y: Integer) of object;

property OnDrawCheckpoint: TRVDrawCheckpointEvent;
```

(introduced in v1.9)

This method informs when some item with *checkpoint* is drawn. It's similar to OnDrawHyperlink 508.

Parameters:

RVData – document containing item.

ItemNo – index of checkpoint's item inside this document.

(**X**,**Y**) – top left corner of this item on page (coordinate of the top left corner of the document is (0,0)).

This is not a custom drawing event but an informational event.

For example, this method can be used to create links inside document in generated PDF files (note: PDF generation requires other third-party components).

1.4.2.4.3.2 TRVReportHelper.OnDraw Hyperlink

Occurs when the ReportHelper draws a hyperlink

```
type
  TRVDrawHyperlinkEvent =
    procedure (Sender: TRVReportHelper;
      RVData: TCustomRVData[608]; ItemNo: Integer;
      R: TRect) of object;

property OnDrawHyperlink: TRVDrawHyperlinkEvent;
```

(introduced in v1.8)

This method informs when some hypertext item is drawn.

Parameters:

RVData – document containing hyperlink.

ItemNo – index of hypertext item inside this document.

R – hyperlink rectangle (coordinate of the top left corner of the document is (0,0)).

This is not a custom drawing event but an informational event.

See also:

- OnDrawCheckpoint[507].

1.4.2.4.3.3 TRVReportHelper.OnGetPageNumber

Request a page number for displaying in "page number" fields[109].

```
type
  TRVGetPageNumberEvent = procedure (Sender: TRVReportHelper[503]; PageNo: Integer;
    var DispPageNo: Integer) of object;;

property OnGetPageNumber: TRVGetPageNumberEvent;
```

(introduced in v15)

Parameters:

PageNo – a number of RVReportHelper's page (from 1 to PagesCount[516]), where the "page number" field is located.

DispPageNo – a number that will be displayed in this field. Initially, it is equal to **PageNo**+PageNoFromNumber[516]-1. You can assign another value to this parameter.

See also:

- UpdatePageNumbers[507]

1.4.3 Components Ancestor Classes

1.4.3.1 TRVScroller

This class is an ancestor of all visual components in TRichView family (TRichView[128], TRichViewEdit[282], TDBRichView[373], TDBRichViewEdit[282], TRVPrintPreview[384]).

Unit RVScroll;

Syntax

```
TCustomRVControl = class (TCustomControl)
TRVScroller = class(TCustomRVControl)
```

Hierarchy

TObject
TPersistent
TComponent
TControl
TWinControl
TCustomControl
TCustomRVControl

Main Features

This is a control what can scroll its content.

Due to limitation of scrollbar ranges, its vertical scrollbar units are not pixels but special units, "RVSU" (10 pixels by default).

Do not use this class directly in your program. Use its descendants, such as TRichView[128].

TRVScroller is not registered as a component itself. It introduces some properties and methods used in descendant components.

Main Properties and Methods

The most important properties of TRVScroller are:

- VScrollPos[512] –- position of vertical scrollbar, measured in "RVSU";
- VScrollMax[512] (read-only) –- maximum value of VScrollPos;
- NoHScroll[511] –- disallows horizontal scrolling;
- NoVScroll[511] –- disallows vertical scrolling;
- HScrollPos[510] –- position of horizontal scrollbar, measured in pixels;
- HScrollMax[509] (read-only) –- maximum value of HScrollPos;
- WheelStep[512] defines speed of mouse wheel scrolling;
- InplaceEditor[510] –- returns "inplace" editor of some item in document (used in tables for editing cells).

The most important method is:

- ScrollTo[513], scrolls content of the component vertically to the specified position (in pixels).

1.4.3.1.1 Properties

1.4.3.1.1.1 TRVScroller.BorderStyle

Determines whether the control has border.

```
type TBorderStyle = bsNone..bsSingle;

property BorderStyle: TBorderStyle;
```

Set BorderStyle to specify whether the control should be outlined.

Value	Meaning
bsNone	No visible border
bsSingle	Border, 3D or themed[511]

1.4.3.1.1.2 TRVScroller.HScrollMax

Maximal possible value for HScrollPos[510]

```
property HScrollMax: Integer;
```

See also properties:

- HScrollPos 510;
- VScrollMax 512.

See also:

- Scrolling in RichView 46

1.4.3.1.1.3 TRVScroller.HScrollPos

Position of horizontal scrollbar. This value is measured in pixels. You can use this property to scroll document to specified horizontal position.

```
property HScrollPos: Integer;
```

This value is in range 0..HScrollMax 509.

See also properties:

- VScrollPos 512.

See also:

- Scrolling in RichView 46.

1.4.3.1.1.4 TRVScroller.HScrollVisible

Set to *False* to hide horizontal scrollbar

```
property HScrollVisible: Boolean;
```

(introduced in version 1.4)

Default value:

True

See also properties:

- VScrollVisible 512.

See also:

- Scrolling in RichView 46.

1.4.3.1.1.5 TRVScroller.InplaceEditor

This property provides access to editor created for editing one of document items.

```
property InplaceEditor: TWinControl;
```

(introduced in version 1.4)

Inplace editor can be used to edit some cell in a table 528 (see "Cell Editing 117"). In future versions, inplace editors can be used for some other types of items.

Inplace editors are derived from TCustomRichViewEdit 282 type.

Inplace editor in its order can have its own inplace editor, and so on. If there is no inplace editor, this property is *nil*. At any time the control can have no more than one inplace editor.

Example: (obtaining top level editor)

```
var
  rve: TCustomRichViewEdit;
...
  rve := MyRichViewEdit;
  while rve.InplaceEditor<>nil do
    rve := rve.InplaceEditor as TCustomRichViewEdit;
  // Now rve points to the top level inplace editor
  // (or MyRichViewEdit itself, if there are no
  // inplace editors).
```

See also:

- TCustomRichViewEdit.TopLevelEditor 295.

1.4.3.1.1.6 TRVScroller.NoVScroll

Disables vertical and/or horizontal scrolling

```
property NoVScroll: Boolean;
property NoHScroll: Boolean;
```

(introduced in versions 11, 15)

Default value

False

1.4.3.1.1.7 TRVScroller.Tracking

Determines whether the control is updated when the user drags the thumb tab of scrollbar.

```
property Tracking: Boolean;
```

Repainting while the thumb tab is dragged can provide useful feedback to the user about how far the control has scrolled, but slows performance.

Default value:

True

1.4.3.1.1.8 TRVScroller.UseXPThemes

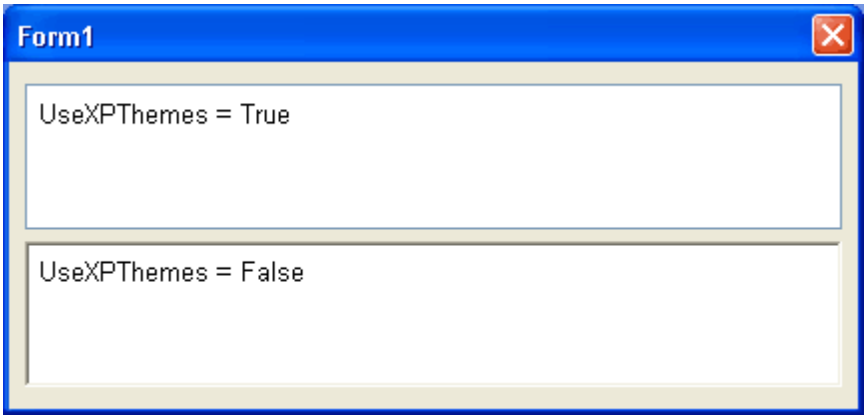
Allows using Windows themes (visual styles)

```
property UseXPThemes: Boolean
```

(introduced in version 1.7)

Themes (visual styles) were introduced in Windows XP and available in newer versions of Windows as well (Windows Vista, 7, 8, 10, etc.)

If this property is set to *True*, the control will use themes if they are available.



UseXPThemes Example

See also:

- BorderStyle 509.

1.4.3.1.1.9 TRVScroller.VScrollMax

Maximal possible value for VScrollPos [512]

```
property VScrollMax: Integer;
```

See also properties:

- VScrollPos [512];
- HScrollMax [509].

See also methods:

- ScrollTo [513].

See also:

- Scrolling in RichView [46].

1.4.3.1.1.10 TRVScroller.VScrollPos

Position of vertical scrollbar. This value is measured in "RVSU" [46], not in pixels because of Windows limitation on scrollbar scrolling range. You can use this property to scroll document to specified position.

```
property VScrollPos: Integer;
```

This value is in range 0..VScrollMax [512].

See also properties:

- HScrollPos [510].

See also methods:

- ScrollTo [513].

See also:

- Scrolling in RichView [46].

1.4.3.1.1.11 TRVScroller.VScrollVisible

Set to *False* to hide vertical scrollbar

```
property VScrollVisible: Boolean;
```

Known problem: scrollbars do not reappear correctly in TRichViewEdit [282] if set VScrollVisible from *False* to *True* when document is not empty.

Default value:

True

See also properties:

- HScrollVisible [510].

See also:

- Scrolling in RichView [46].

1.4.3.1.1.12 TRVScroller.WheelStep

Defines how much the document will be scrolled when the user turns a mouse wheel.

```
property WheelStep: Integer;
```

If set to 0, the control ignores mouse wheel messages (they will be sent to the parent control).

Mouse wheel scrolling in TRVScroller depends on the system settings. The system suggests the default scrolling speed. If the suggested value is "one page", this property is ignored (if nonzero). If the suggested value is "no scrolling", no scrolling occurs. Otherwise, the component is scrolled by Round(VSmallStep [159] *WheelStep*suggested/3) pixels.

Default value:

2

1.4.3.1.2 Methods

1.4.3.1.2.1 TRVScroller.Create

A constructor, creates and initializes a TRVScroller instance.

```
constructor Create(AOwner: TComponent);override;
```

Do not create TRVScroller objects in your applications directly. Create components inherited from it:

- TRichView 128;
- TRichViewEdit 282;
- TDBRichView 373;
- TDBRichViewEdit 282;
- TRVPrintPreview 384.

1.4.3.1.2.2 TRVScroller.ScrollTo

Scrolls content of the document vertically to the specified position (in pixels).

```
procedure ScrollTo(y: Integer);
```

Vertical scrollbar units are not pixels, so scrolling will be not exactly to specified position (but a little bit higher).

See also properties:

- VScrollPos 512;
- VScrollMax 512.

See also:

- Scrolling in RichView 46.

1.4.3.1.3 Events

1.4.3.1.3.1 TRVScroller.OnHScrolled

Occurs when the document is scrolled horizontally

```
property OnHScrolled: TNotifyEvent;
```

See also events:

- OnVScrolled 513.

See also:

- Scrolling in RichView 46.

1.4.3.1.3.2 TRVScroller.OnVScrolled

Occurs when document is scrolled vertically

```
property OnVScrolled: TNotifyEvent;
```

See also events:

- OnHScrolled 513.

See also properties of TRichView:

- FirstItemVisible 144;
- LastItemVisible 144.

See also:

- Scrolling in RichView 46.

1.4.3.2 TCustomRVPrint

TCustomRVPrint is an Ancestor of TRVPrint[479] and TRVReportHelper[503] components.

This is an intermediate class. Do not use it directly in your programs.

Unit PtblRV;

Syntax

```
TCustomRVPrint = class(TComponent)
```

Hierarchy

TObject
TPersistent
TComponent

1.4.3.2.1 Properties

1.4.3.2.1.1 TCustomRVPrint.ColorMode

Sets color mode for printing and drawing preview

```
property ColorMode: TRVColorMode[625];
```

(Introduced in version 1.8)

Default values:

- *rvcmColor* for TRVReportHelper[503];
- *rvcmPrinterColor* for TRVPrint[479].

1.4.3.2.1.2 TCustomRVPrint.EndAt

Height of document on the last page, in device pixels.

```
property EndAt: Integer;
```

This property does not include margins.

You can use this property only when document is completely formatted (see FormatPages[490] for TRVPrint, Init[507] and FormatNextPage[506] for TRVReportHelper).

For TRVReportHelper, you can use GetLastPageHeight[506] instead (it's more convenient, because it includes top and bottom margins).

See also properties:

- StartAt[517].

1.4.3.2.1.3 TCustomRVPrint.FirstPageNo

Returns the index of the first page that will be printed, from 1.

```
property FirstPageNo: Integer;
```

(Introduced in version 13)

This property takes the properties MinPrintedItemNo[516] and MaxPrintedItemNo[515] into account. By default, when these properties define the range including the whole document, FirstPageNo returns 1. If MinPrintedItemNo[516]>0, this property returns the index of the page containing the MinPrintedItemNo[516]-th item (see GetPageNo[518]). If MinPrintedItemNo[516]>=ItemCount, FirstPageNo returns PagesCount[516]+1 (and nothing will be printed).

1.4.3.2.1.4 TCustomRVPrint.IgnorePageBreaks

Allows ignoring page breaks in a document.

```
property IgnorePageBreaks: Boolean;
```

(Introduced in version 14)

If *True*, the component ignores the following properties when printing:

- TRichView.PageBreaksBeforeItems[149][];
- TRVTableItemInfo.Rows[].PageBreakBefore[578].

This property is useful in TRVReportHelper[503] component, if you want to display the loaded document completely in a single rectangle.

Default value

False

1.4.3.2.1.5 TCustomRVPrint.LastPageNo

Returns the index of the last page that will be printed, from 1.

```
property LastPageNo: Integer;
```

(Introduced in version 13)

This property takes the properties MinPrintedItemNo[516] and MaxPrintedItemNo[515] into account. By default, when these properties define the range including the whole document, LastPageNo returns PagesCount[516]. If MaxPrintedItemNo[515] is inside the items range (0..ItemCount-1), this property returns the index of the page containing the MaxPrintedItemNo[515]-th item (see GetPageNo[518]).

1.4.3.2.1.6 TCustomRVPrint.MaxPrintedItemNo

Defines the index of the last item to print.

```
property MaxPrintedItemNo: Integer;
```

(Introduced in version 13)

The component prints items from MinPrintedItemNo[516] to MaxPrintedItemNo. By default, this range includes all items.

The value -1 is treated like ItemCount-1, i.e. the document is printed to the end.

Items outside of this range are not printed (but they still occupy their spaces). The pages that do not contain items in this range are skipped (the component prints pages from FirstPageNo[514] to LastPageNo[515]).

These properties allow implementing an incremental printing: when the next portion of the document is ready, it can be printed below the previously printed fragment. This feature can be used for printing accounting journals, logs, etc.

Default value:

-1

1.4.3.2.1.7 TCustomRVPrint.MetafileCompatibility

Specifies whether printing output is compatible with metafiles.

```
property MetafileCompatibility: Boolean;
```

(introduced in version 16)

If RVStyle.TextEngine[403] = *rvteUniscribe*, or for bidirected text, the component may draw/print text using glyph indexes in fonts instead of character codes. When printing output is captured to a metafile, this metafile can be viewed correctly only on computers having the same font installed. On computers without this font, or with different version of this font, such metafiles may look completely incorrect. Also, it may cause problems for some virtual printers.

You can set **MetafileCompatibility**=*True* to switch to metafile-compatible output (less efficient, and in some cases of a lower quality).

Default value

False

1.4.3.2.1.8 TCustomRVPrint.MinPrintedItemNo

Defines the index of the first item to print.

```
property MinPrintedItemNo: Integer;
```

(Introduced in version 13)

The component prints items from MinPrintedItemNo to MaxPrintedItemNo[515]. By default, this range includes all items.

Items outside of this range are not printed (but they still occupy their spaces). The pages that do not contain items in this range are skipped (the component prints pages from FirstPageNo[514] to LastPageNo[515]).

These properties allow implementing an incremental printing: when the next portion of the document is ready, it can be printed below the previously printed fragment. This feature can be used for printing accounting journals, logs, etc.

Default value:

0

1.4.3.2.1.9 TCustomRVPrint.PageNoFromNumber

Specifies the number of the first page for "page number" fields[109].

```
property PageNoFromNumber: Integer;
```

(introduced in version 15)

Default value:

1

See also:

- TRVReportHelper.OnGetPageNumber[508]

1.4.3.2.1.10 TCustomRVPrint.PagesCount

Total number of pages in the document.

```
property PagesCount: Integer;
```

You can use this property only when document is formatted (see FormatPages[490] for TRVPrint, Init[507] and FormatNextPage[506] for TRVReportHelper).

1.4.3.2.1.11 TCustomRVPrint.PreviewCorrection

Determines whether the component will make correction when drawing print preview.

```
property PreviewCorrection: Boolean;
```

This property does not affect to the printing itself, only to the preview.

Due to different resolutions on the screen and on the printer, print preview may be inaccurate (because text not always can be scaled proportionally).

If this property is *True*, the component tries to perform some corrections to improve preview quality (for example, it adjusts intercharacter spacing).

Default value:

True

1.4.3.2.1.12 TCustomRVPrint.StartAt

Offset from the top of the first page (from the top margin) to the beginning of document on it, in device pixels.

```
StartAt: Integer;
```

This property is useful (in conjunction with EndAt [514]) in TRVPrint.ContinuousPrint [488] method.

Default value:

0

1.4.3.2.1.13 TCustomRVPrint.TransparentBackground

Allows to turn off printing of the document background

```
property TransparentBackground: Boolean;
```

If set to *True*, background will not be printed.

Default value:

False

See also:

- *rvtoWhiteBackground* option in PrintOptions [542] of tables.

1.4.3.2.2 Methods

1.4.3.2.2.1 TCustomRVPrint.Clear

Clears memory allocated for printing (in FormatPages [490])

```
procedure Clear;
```

After calling Clear, TRVPrint will not be able to print and display preview until the next call of AssignSource [488] and FormatPages [490] (or Init [507] and FormatNextPage [506] in TRVReportHelper).

1.4.3.2.2.2 TCustomRVPrint.GetFirstItemOnPage

Returns index of the first item on the page

```
procedure GetFirstItemOnPage(PageNo: Integer; var ItemNo, OffsetInItem: Integer);
```

Input parameter:

PageNo – number of page (from 1).

Output parameters:

ItemNo – index of the first item on the page (from 0)

OffsetInItem:

- for non-text items: 0;
- for text items: index of the first character on the page (from 1)

Note: this information is not enough when using tables [528]. Tables can be printed on several pages, and using this information you cannot determine which part of table starts the page.

See also:

- GetPageNo [518].

See also properties:

- PagesCount [516].

1.4.3.2.2.3 TCustomRVPrint.GetPageNo

Returns index of page containing the specified place of the document.

```
function GetPageNo(RVData: TCustomRVData; ItemNo, OffsetInItem: Integer): Integer;
```

Parameters:

The parameters define a position in the source document,

RVData is a document/subdocument. It can be either RVData[151] of the source TRichView control, or table cell, or RVData[151] of a cell inplace editor.

For TRVPrint[479], the source TRichView control is a control assigned by AssignSource[488]. For TRVReportHelper, the source TRichView control is RichView[505] property.

ItemNo is the index of item in this **RVData**.

OffsetInItem is the position inside this item. For non-text items, 0 is a position before the item, 1 is a position after the item. For text items[88], offsets are counted from 1 (1 - before the first character, 2 - before the second character, ... Length+1 - after the last character).

Return value:

Index of the page for this position, from 1.

See also:

- GetFirstItemOnPage[517].

See also methods of TCustomRichView:

- GetOffsAfterItem[200];
- GetOffsBeforeItem[201].

1.4.3.2.2.4 TCustomRVPrint.UpdatePaletteInfo

Call this method when the screen color resolution changed (on WM_DISPLAYCHANGE), if you are using this RVPrint for displaying print preview.

This method is necessary (only) if you need to support 256 color display mode.

```
procedure UpdatePaletteInfo;
```

Example[143]

TRVPrint uses the same method of working in 256 color mode as its source RichView (see RichView.DoInPaletteMode[142]). Information about source RichView.DoInPaletteMode is updated when executing AssignSource[488] method.

It's not necessary to call this method if RichView.DoInPaletteMode=*rvpaDoNothing,* or if you are not using this TRVPrint object for displaying print preview in TRVPrintPreview[384].

Call UpdatePaletteInfo if you created TRVPrint at run-time (instead of placing at design-time on a form) after you have created it.

TRVPrintPreview controls do not have public UpdatePaletteInfo method. They use information from the linked RVPrint. They update their palettes when you assign their PageNo[522] or RVPrint[384] properties.

See also methods of TRichView:

- UpdatePaletteInfo[238].

1.4.3.2.3 Events

1.4.3.2.3.1 TCustomRVPrint.OnPrintComponent

Occurs when printing inserted control [94]

```
property OnPrintComponent: TRVPrintComponentEvent;

TRVPrintComponentEvent = procedure(Sender: TCustomRVPrint [514];
  PrintMe: TControl; var ComponentImage: TBitmap) of object;
```

If you are not satisfied how RichView prints some classes of controls, you can print them yourself.

This event requests image of control, in screen resolution, in bitmap.

Input parameters:

PrintMe – the control for printing.

ComponentImage – *nil*.

Output parameter:

ComponentImage – image of **PrintMe**. You should create this bitmap in this event and should not destroy it yourself (RVPrint will do it for you).

Create the bitmap having width and height of PrintMe (you can return larger bitmap and it will be stretched, allowing to create better print output).

There are several methods for creating bitmap for components in **CtrlImage** unit: DrawControl and others [616]. The component use them for default controls printing.

Example:

```
procedure MyForm.MyRVPrintPrintComponent(Sender: TCustomRVPrint;
  PrintMe: TControl; var ComponentImage: TBitmap);
var r: TRect;
    State: Cardinal
begin
  if PrintMe is TImage then begin
    // printing TImage
    ComponentImage := TBitmap.Create;
    ComponentImage.Assign(TImage(PrintMe).Picture.Graphic);
    end
  else if PrintMe is TCheckBox then begin
    // printing TCheckBox
    ComponentImage := TBitmap.Create;
    ComponentImage.Width := PrintMe.Width;
    ComponentImage.Height := PrintMe.Height;
    ComponentImage.Canvas.Brush.Color := TCheckBox(PrintMe).Color;
    r := Rect(0,0, PrintMe.Width, PrintMe.Height);
    ComponentImage.Canvas.FillRect(r);
    if TCheckBox(PrintMe).Checked then
      State := DFCS_CHECKED
    else
      State := 0;
    r.Right := r.Bottom;
    DrawFrameControl(ComponentImage.Canvas.Handle, r, DFC_BUTTON,
      DFCS_BUTTONCHECK or State);
    r := Rect(r.Right,0, PrintMe.Width, PrintMe.Height);
    ComponentImage.Canvas.Font := TCheckBox(PrintMe).Font;
    DrawText(ComponentImage.Canvas.Handle, PChar(' '+TCheckBox(PrintMe).Caption), -1, r,
      DT_VCENTER or DT_SINGLELINE);
```

```
    end
  else
    // printing other controls
    ComponentImage := DrawControl(PrintMe)
end;
```

In TRVPrint, this event is generated by:

- Print;
- PrintPages;
- MakePreview;
- MakeScaledPreview;
- DrawPreview;
- when drawing TRVPrintPreview component.

1.4.3.3 TCustomRVPrintPreview

TCustomRVPrintPreview is an ancestor of TRVPrintPreview component. This class draws a blank page, introduces mechanisms of page switching and scaling.

Unit CRVPP;

Syntax

```
TCustomRVPrintPreview = class(TRVScroller)
```

Hierarchy

TObject
TPersistent
TComponent
TControl
TWinControl
TCustomControl
TRVScroller

Cursors

Print preview uses two cursors:

- ZoomInCursor,
- ZoomOutCursor.

Scaling

Scaling is defined by:

- ZoomMode property – mode of scaling;
- ZoomPercent property – scaling percent;
- SetZoom method changes ZoomPercent;
- OnZoomChanged event occurs when ZoomPercent changes;
- ClickMode property allows/disallows zooming.

Page switching

First, Last, Prev, Next methods, PageNo property.

Displaying margins

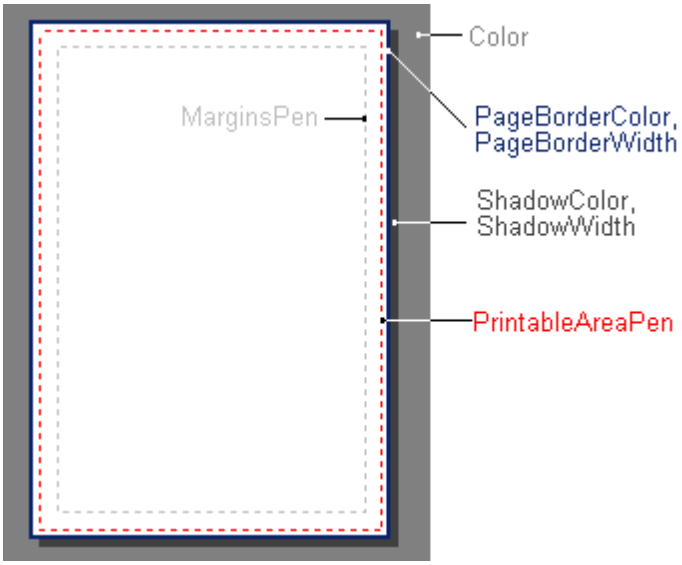
MarginsPen property

Colors and widths

- Color – background color;
- PageBorderColor, PageBorderWidth define border around the page;

- ShadowColor[523], ShadowWidth[523] define shadow.

Scheme



TCustomRVPrintPreview

1.4.3.3.1 Properties

1.4.3.3.1.1 TCustomRVPrintPreview.ClickMode

Defines the action occurring when the user clicks the preview.

```
property ClickMode: TRVClickMode;

type
  TRVClickMode = (rvcmNone, rvcmSwitchZoom);
```

(introduced in v1.8)

Click Mode	Meaning
rvcmNone	Clicking does nothing, zoom-in[523] and zoom-out[524] cursors are not used.
rvcmSwitchZoom	Clicking switches between 100% and custom zoom.

Default value:

rvcmSwitchZoom

1.4.3.3.1.2 TCustomRVPrintPreview.Color

Background color

```
property Color: TColor;
```

(introduced in v1.8)

Note: In Delphi XE2+, if this value contains a system color, and a custom style is applied to the application, the component uses the corresponding style color.

Default value:

clBtnShadow

See also:

- Picture at the bottom of TCustomRVPrintPreview[520] topic.

1.4.3.3.1.3 TCustomRVPrintPreview.MarginsPen

This pen is used for drawing rectangle around the document area on the page preview.

```
property MarginsPen: TRVMarginsPen;
```

By default, MarginsPen.Style=*psClear*, so the rectangle is invisible:

```
type
  TRVMarginsPen = class (TPen)
    property Style default psClear;
    property Color default clSilver;
  end;
```

See also properties:

- PrintableAreaPen[523].

See also properties of TRVPrint[479]:

- Margins[486],

See also:

- Picture in TRVPrint topic[479].

1.4.3.3.1.4 TCustomRVPrintPreview.PageBorderColor

Color of frame around the page.

```
property PageBorderColor: TColor;
```

(introduced in v1.8)

Note: In Delphi XE2+, if this value contains a system color, and a custom style is applied to the application, the component uses the corresponding style color.

Default value:

clHighlight

See also:

- Picture at the bottom of TCustomRVPrintPreview[520] topic.

1.4.3.3.1.5 TCustomRVPrintPreview.PageBorderWidth

Width of frame around the page, pixels

```
property PageBorderWidth: Integer;
```

(introduced in v1.8)

Default value:

2

See also:

- Picture at the bottom of TCustomRVPrintPreview[520] topic.

1.4.3.3.1.6 TCustomRVPrintPreview.PageNo

Index of displayed page (from 1)

```
property PageNo: Integer;
```

See also methods:

- Prev[526];
- Next[526];
- First[525];

- Last [525].

1.4.3.3.1.7 TCustomRVPrintPreview.PrintableAreaPen

This pen is used for drawing rectangle around the printable area on the page preview.

```
property PrintableAreaPen: TRVPAPen;
```

(introduced in version 10)

By default, PrintableAreaPen.Style=*psClear*, so the rectangle is invisible:

```
type
  TRVPAPen = class (TPen)
    property Style default psClear;
    property Color default clRed;
  end;
```

Printer cannot print beyond the area shown on this rectangle, see WinAPI function GetDeviceCaps, PHYSICAL*** constants.

See also properties:

- MarginsPen [522].

See also:

- Picture in TRVPrint topic [479].

1.4.3.3.1.8 TCustomRVPrintPreview.Shadow Color

Color of the page shadow.

```
property ShadowColor: TColor;
```

(introduced in v1.8)

Note: In Delphi XE2+, if this value contains a system color, and a custom style is applied to the application, the component uses the corresponding style color.

Default value:

cl3DDkShadow

See also:

- Picture at the bottom of TCustomRVPrintPreview [520] topic.

1.4.3.3.1.9 TCustomRVPrintPreview.Shadow Width

Width of the page shadow, pixels.

```
property ShadowWidth: TColor;
```

(introduced in v1.8)

Default value:

20

See also:

- Picture at the bottom of TCustomRVPrintPreview [520] topic.

1.4.3.3.1.10 TCustomRVPrintPreview.ZoomInCursor

Cursor appeared when the current scale<100%, or the current scale=100% switched from the scale>100%

```
property ZoomInCursor: TCursor;
```

If ZoomPercent [525]<>100%, clicking on component changes ZoomPercent [525] to 100%.

If ZoomPercent [525]=100%, clicking on component changes ZoomPercent [525] to previous scale.

Default value:

 *crRVZoomIn* (102)

(this cursor is registered automatically)

See also properties:

- ZoomOutCursor 524;
- ZoomPercent 525.

See also:

- RichView cursors 77.

1.4.3.3.1.11 TCustomRVPrintPreview.ZoomMode

Scaling mode

```
type // defined in RVStyle unit
  TRVZoomMode = (rvzmFullPage, rvzmPageWidth, rvzmCustom);
property ZoomMode : TRVZoomMode;
```

Assigning value to this property has an immediate effect.

Zoom Mode	Meaning
rvzmFullPage	Changing ZoomPercent 525 to display full page, keeping the full-page view during resizing (changing ZoomPercent 525 when needed)
rvzmPageWidth	Changing ZoomPercent 525 to fit width of page, keeping the page-width view during resizing (changing ZoomPercent 525 when needed)
rvzmCustom	Scaling according to the ZoomPercent 525 value

Default value:

rvzmCustom

1.4.3.3.1.12 TCustomRVPrintPreview.ZoomOutCursor

Cursor appeared when the current scale>100%, or the current scale=100% switched from the scale<100%

```
property ZoomOutCursor: TCursor;
```

If ZoomPercent 525<>100%, clicking on component changes ZoomPercent 525 to 100%.

If ZoomPercent 525=100%, clicking on component changes ZoomPercent 525 to previous scale.

Default value:

 *crRVZoomOut* (103)

(this cursor is registered automatically)

See also properties:

- ZoomInCursor 523;
- ZoomPercent 525.

See also:

- RichView cursors 77.

1.4.3.3.1.13 TCustomRVPrintPreview.ZoomPercent

Scale of preview, %

```
property ZoomPercent: Integer;
```

Although assigning to this property is possible, the preferable way to change preview scale is calling SetZoom [526] method.

This property is changed automatically when resizing the component, if ZoomMode [524] = *rvzmFullPage* or *rvzmPageWidth*.

It's recommended to keep this property in 10%–500% range.

See also methods:

- SetZoom [526].

See also properties:

- ZoomMode [524].

See also events:

- OnZoomChanged [526].

1.4.3.3.2 Methods

1.4.3.3.2.1 TCustomRVPrintPreview.Create

A constructor, creates a new TCustomRVPrintPreview object.

```
constructor Create(AOwner: TComponent);override;
```

Do not to create TCustomRVPrintPreview objects, create TRVPrintPreview [384] components instead.

1.4.3.3.2.2 TCustomRVPrintPreview.First

Switches the current page (PageNo [522] property) to the first page.

```
procedure First;
```

See also properties:

- PageNo [522].

See also methods:

- Prev [526];
- Next [526];
- Last [525].

1.4.3.3.2.3 TCustomRVPrintPreview.Last

Switches the current page (PageNo [522] property) to the last page.

```
procedure Last;
```

See also properties:

- PageNo [522];
- TRVPrintPreview.RVPrint [384].PagesCount [516].

See also methods:

- Prev [526];
- Next [526];
- First [525].

1.4.3.3.2.4 TCustomRVPrintPreview.Next

Switches the current page (PageNo 522 property) to the next page, if possible.

```
procedure Next;
```

See also properties:

- PageNo 522;
- TRVPrintPreview.RVPrint 384.PagesCount 516.

See also methods:

- Prev 526;
- First 525;
- Last 525.

1.4.3.3.2.5 TCustomRVPrintPreview.Prev

Switches the current page (PageNo 522 property) to the previous page, if possible.

```
procedure Prev;
```

See also properties:

- PageNo 522.

See also methods:

- Next 526;
- First 525;
- Last 525.

1.4.3.3.2.6 TCustomRVPrintPreview.SetZoom

Changes ZoomMode 524 to *rvzmCustom* and ZoomPercent 525 to **Percent**.

```
procedure SetZoom(Percent: Integer);
```

Calling this method has an immediate effect.

It's recommended to keep **Percent** in 10%–500% range.

See also properties:

- ZoomMode 524;
- ZoomPercent 525.

1.4.3.3.3 Events

1.4.3.3.3.1 TCustomRVPrintPreview.OnZoomChanged

Occurs when ZoomPercent 525 changes.

```
property OnZoomChanged: TNotifyEvent;
```

1.5 Item Types

TCustomRVItemInfo is a basic type for all items 85 in TRichView documents.

Detailed description of all classes for item types is beyond the scope of this help file.

You can add, access and modify all items using methods of TRichView 128 and TRichViewEdit 282 instead of working with the item classes directly, so details of implementation are hidden from programmers. The exceptions are:

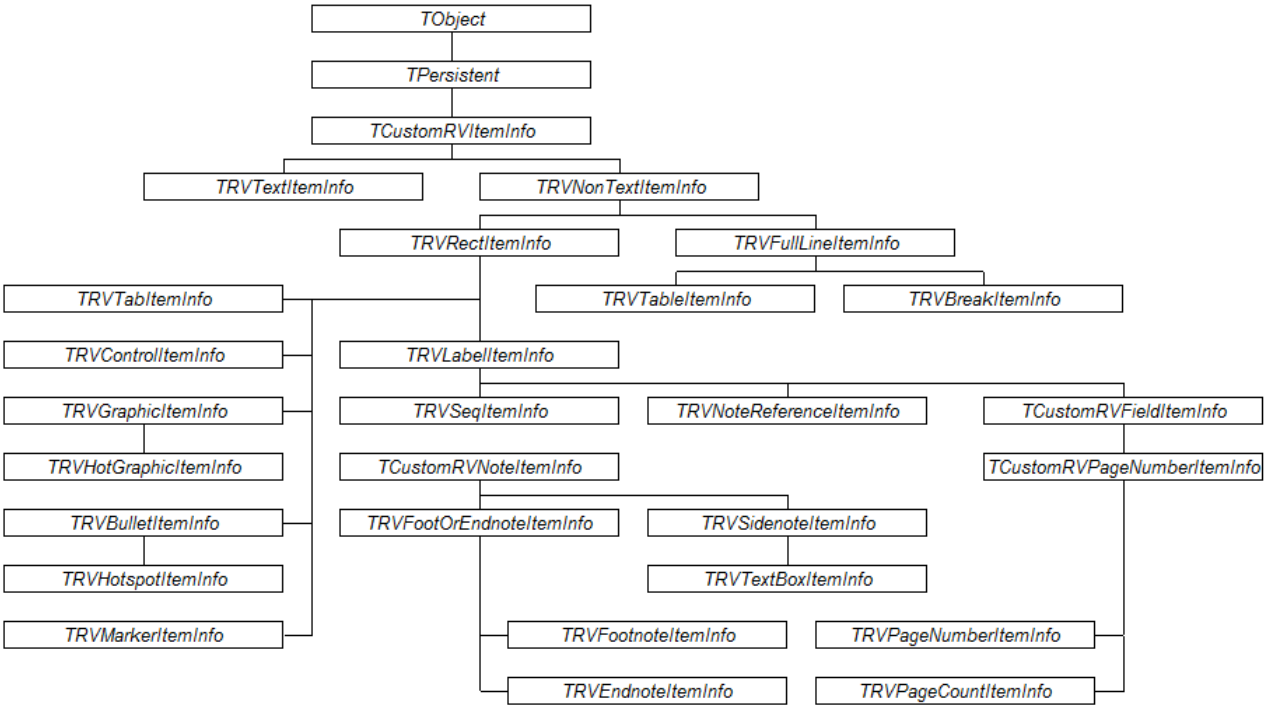
- TRVTableItemInfo 528;
- TRVLabelItemInfo 579 and classes, inherited from it.

Properties of tables must be accessed and modified directly. Properties of labels (and inherited classes) can be

modified either directly or using methods of TRichView and TRichViewEdit.

Class	Unit	Meaning
TCustomRVItemInfo	RVItem	Basic class for TRichView items.
TRVTextItemInfo	RVItem	Text item 88.
TRVNonTextItemInfo	RVItem	Basic class for non-text items.
TRVRectItemInfo	RVItem	Basic class for rectangular items.
TRVFullLineItemInfo	RVItem	Basic class for items occupying the full line.
TRVTabItemInfo	RVItem	Tabulator 89.
TRVControlItemInfo	RVItem	Item containing Delphi control 94.
TRVGraphicItemInfo	RVItem	Picture 90.
TRVHotGraphicItemInfo	RVItem	*Hot-picture* 93 (picture-hyperlink).
TRVBulletItemInfo	RVItem	*Bullet* 96 (image from TImageList).
TRVHotspotItemInfo	RVItem	*Hotspot* 97 (*bullet*-hyperlink).
TRVMarkerItemInfo	RVMarker	Paragraph marker 99.
TRVTableItemInfo 528	RVTable	Table 98.
TRVBreakItemInfo	RVItem	*Break* 93 (horizontal line).
TRVLabelItemInfo 579	RVLabelItem	Label 101 (non-text item looking like text).
TRVSeqItemInfo 582	RVSeqItem	Numbered sequence 102.
TCustomRVNoteItemInfo 586	RVNote	Basic class for notes and text boxes.
TRVFootOrEndnoteItemInfo 586	RVNote	Basic class for footnotes and endnotes
TRVEndnoteItemInfo 587	RVNote	Endnote 103.
TRVFootnoteItemInfo 589	RVNote	Footnote 104.
TRVSidenoteItemInfo 591	RVSidenote	Sidenote 105
TRVTextBoxItemInfo 601	RVSidenote	Text box 107
TRVNoteReferenceItemInfo 602	RVNote	Reference to the parent note 108
TRVPageNumberItemInfo 605	RVFieldItems	"Page number" field 109
TRVPageCountItemInfo 607	RVFieldItems	"Page count" field 110

Hierarchy



Other Types

```
TCustomRVItemInfoClass = class of TCustomRVItemInfoClass;
```

1.5.1 Table - TRVTableItemInfo

TRVTableItemInfo is a class representing table [98] in TRichView documents. This is not a component. Objects of this class are created at runtime and appended to TRichView (see AddItem [166]) or inserted into TRichViewEdit (see InsertItem [320]). Style [194] of this item type: *rvsTable* (-60)

Unit RVTable.

Syntax

```
TRVTableItemInfo = class (TRVFullLineItemInfo)
```

Hierarchy

TObject
TPersistent
TCustomRVItemInfo [526]
TRVNonTextItemInfo
TRVFullLineItemInfo

Overview

See overview topics about tables [111].

Properties

The most of properties of this class are published. These properties can be saved/loaded in RVF [59] files or streams.

The main property of TRVTableItemInfo is Cells [537]. The number of rows is RowCount [543], number of columns is

ColCount 538.

Properties of rows are available via Rows 543 property.

Width of table can be defined using BestWidth 531 property.

Properties defining border around the table:

- BorderWidth 534;
- BorderStyle 533;
- BorderColor 532;
- BorderLightColor 532;
- VisibleBorders 544;
- also BorderHSpacing 532, BorderVSpacing 533.

Properties defining default border around cells:

- CellBorderWidth 535;
- CellBorderStyle 535;
- CellBorderColor 534;
- CellBorderLightColor 534.

Properties defining default colors of cells in rows and columns:

- HeadingRowColor 546, LastRowColor, FirstColumnColor, LastColumnColor, EvenColumnsColor, OddRowsColor, OddColumnsColor, EvenRowsColor

Properties defining spacing between cells:

- CellHSpacing 536;
- CellVSpacing 537;
- BorderHSpacing 532;
- BorderVSpacing 533;
- also CellHPadding 536, CellVPadding 537.

Background properties:

- Color 538;
- Opacity 540;
- BackgroundImage 530;
- BackgroundStyle 531;
- BackgroundImageFileName 531.

Border colors:

- BorderColor 532;
- BorderLightColor 532;
- CellBorderColor 534;
- CellBorderLightColor 534.

Rules:

- HRuleWidth 540;
- HRuleColor 539;
- HOutermostRule 539;
- VRuleWidth 545;
- VRuleColor 545;
- VOutermostRule 545.

Export:

- TextColSeparator 543;
- TextRowSeparator 544;
- BackgroundImageFileName 531.

Other properties:

- Options 540;
- PrintOptions 542;
- HeadingRowCount 539.

Methods

See operations on tables [119].

Events

Tables are not components, so events cannot be assigned at design time in the Object Inspector. They must be assigned from code, when the table is created.

Events cannot be saved with tables in RVF [59] files or streams. Reassign them when tables are loaded, use TRichView.OnItemAction [244] event.

Events:

- OnCellEditing [564];
- OnDrawBorder [565].

Related

- Drawing table grid [392];
- New row auto insertion [657].

1.5.1.1 Properties

1.5.1.1.1 TRVTableItemInfo.BackgroundImage

Background image for the table.

```
property BackgroundImage: TGraphic;
```

(introduced in v1.8)

Position of the image is defined by the BackgroundStyle [533] property.Transparent images are supported.

Assignment to this property copies the image, so you still need to free the source image yourself.

Example 1 (assigning image)

```
bmp := TBitmap.Create;
bmp.LoadFromFile('c:\image.bmp');
table.BackgroundImage := bmp;
bmp.Free;
```

Example 2 (clearing the image)

```
table.BackgroundImage := nil;
```

See also:

- Undo of assignments to table properties.

See also properties:

- BackgroundStyle [531];
- Color [538].

See also properties of table cell:

- BackgroundImage [567];
- BackgroundStyle [568].

See also properties of TRichView:

- BackgroundBitmap [136];
- BackgroundStyle [136].

1.5.1.1.2 TRVTableItemInfo.BackgroundImageFileName

A string storing the file name associated with BackgroundImage[530].

```
property BackgroundImageFileName: String;
```

(introduced in v1.9)

If *rvsoUseItemImageFileNames* is included in the Options parameter of SaveHTML[216]/SaveHTMLEx[217], images with defined (non-empty) file names will not be saved, but their file names will be written in HTML. This value is also accessible as extra string property[630] *rvespImageFileName*.

Assignment to this property cannot be undone by the user[54].

See also properties of table cell:

- BackgroundImageFileName[568].

1.5.1.1.3 TRVTableItemInfo.BackgroundStyle

Defines position of BackgroundImage[530].

```
property BackgroundStyle: TRVItemBackgroundStyle[636];
```

(introduced in v1.8)

See also:

- Undo of assignments to table properties.

See also properties:

- BackgroundImage[530];
- Color[538].

See also properties of table cell:

- BackgroundStyle[568];
- BackgroundImage[567].

See also properties of TRichView:

- BackgroundStyle[136];
- BackgroundBitmap[136].

1.5.1.1.4 TRVTableItemInfo.BestWidth

Preferred width for the table.

```
property BestWidth: TRVHTMLLength[635];
```

This is a preferred width for the table, measured either in Units[404] of the linked TRVStyle[385] component (pixels or twips) or in percent.

The table can be wider than this value if it is required to display its content without overlapping.

If BestWidth is undefined (=0), RichView tries to calculate the table width basing on its cells. In this version RichView can do it only if all columns have widths defined in TRVStyleUnits[644] (have at least one cell with BestWidth[569] defined in TRVStyleUnits[644], and do not have cells with BestWidth[569] defined in percents). Otherwise BestWidth=0 works like BestWidth=-100 (100%).

This property corresponds to HTML <table width> attribute.

See also undo of assignments to table properties.

Default value:

0

See also:

- Table Resizing[124];
- Undo of assignments to table properties.

See also properties of table cell:

- BestWidth[569];
- BestHeight[568].

1.5.1.1.5 TRVTableItemInfo.BorderColor

Defines color of the table border.

```
property BorderColor: TColor
```

This property defines either color of flat border or "dark" color of 3d border around the table.

See BorderStyle[533] for details.

This property corresponds to HTML <table bordercolor> or <table bordercolordark> attributes (specific to Microsoft Internet Explorer).

Default value:

clWindowText

See also:

- Undo of assignments to table properties;
- Colors and layout of tables[114] (with pictures).

See also properties

- BorderLightColor[532];
- CellBorderColor[534];
- CellBorderLightColor[534].

1.5.1.1.6 TRVTableItemInfo.BorderHSpacing

Horizontal spacing between the table border and the outermost cells.

```
property BorderHSpacing: TRVStyleLength[644];
```

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units[404] of the linked TRVStyle[385] component.

BorderHSpacing be saved in HTML+CSS if:

- BorderHSpacing=CellHSpacing[536], or
- BorderHSpacing>=CellHSpacing[536] and CellHSpacing[536]>=0

Default value:

2

Note: this default value assumes that RVStyle.Units[404]=*rvstuPixels*, otherwise it could be too small.

See also:

- Undo of assignments to table propeties;
- Colors and layout of tables[114] (with pictures).

See also properties:

- BorderVSpacing[533];
- CellHSpacing[536];
- CellVSpacing[537].

1.5.1.1.7 TRVTableItemInfo.BorderLightColor

This property defines "light" color of 3d border around the table.

```
property BorderLightColor: TColor;
```

See BorderStyle[533] for details.

This property corresponds to HTML <table bordercolorlight> attribute (specific to Microsoft Internet Explorer).

Default value:

clBtnHighlight

See also:

- Undo of assignments to table propeties;
- Colors and layout of tables [114] (with pictures).

See also properties:

- BorderColor [532];
- CellBorderColor [534];
- CellBorderLightColor [534].

1.5.1.1.8 TRVTableItemInfo.BorderStyle

This property defines style of border around the table.

```
property BorderStyle: TRVTableBorderStyle [645];
```

If you want to hide table border, set BorderWidth [534] = 0.

Default value:

rvtbRaised

See also:

- Undo of assignments to table propeties;
- Colors and layout of tables [114] (with pictures).

See also properties:

- CellBorderStyle [535];
- BorderColor [532];
- BorderLightColor [532];
- BorderWidth [534];
- VisibleBorders [544].

1.5.1.1.9 TRVTableItemInfo.BorderVSpacing

Vertical spacing between border and the outermost cells.

```
property BorderVSpacing: TRVStyleLength [644];
```

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units [404] of the linked TRVStyle [385] component.

BorderVSpacing be saved in HTML+CSS if:

- BorderVSpacing=CellVSpacing [537], or
- BorderVSpacing>=CellVSpacing [537] and CellVSpacing [537]>=0

Default value:

2

Note: this default value assumes that RVStyle.Units [404]=*rvstuPixels*, otherwise it could be too small.

See also:

- Undo of assignments to table propeties;
- Colors and layout of tables [114] (with pictures).

See also properties:

- BorderHSpacing [532];
- CellHSpacing [536];
- CellVSpacing [537].

1.5.1.1.10 TRVTableItemInfo.BorderWidth

Width of border around the table.

```
property BorderWidth: TRVStyleLength[644];
```

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units[404] of the linked TRVStyle[385] component.

Set to 0 to hide border.

This property corresponds to HTML <table border>.

Default value:

0

See also:

- Undo of assignments to table propeties;
- Colors and layout of tables[114] (with pictures).

See also properties:

- CellBorderWidth[535];
- VisibleBorders[544].

1.5.1.1.11 TRVTableItemInfo.CellBorderColor

Defines default border color for all table cells.

```
property CellBorderColor: TColor
```

This property defines either default color of flat border or default "dark" color of 3d border around the table cells.

See CellBorderStyle[535] for details.

This value can be overridden with BorderColor[569] of cell.

This table attribute can be saved in HTML only with use of CSS (Cascading Style Sheet).

Default value:

clWindowText

See also:

- Undo of assignments to table propeties;
- Colors and layout of tables[114] (with pictures).

See also properties:

- CellBorderLightColor[534];
- BorderLightColor[532];
- BorderColor[532].

See also properties of table cell:

- BorderColor[569];
- BorderLightColor[570].

1.5.1.1.12 TRVTableItemInfo.CellBorderLightColor

Defines default "light" color of 3d border for all table cells.

```
property CellBorderLightColor: TColor;
```

See CellBorderStyle[535] for details. To hide cell borders, set CellBorderWidth[535] = 0.

This table attribute can be saved in HTML only with use of CSS (Cascading Style Sheet).

This value can be overridden with BorderLightColor[570] of cell.

Default value:

clBtnHighlight

See also:

- Undo of assignments to table propeties;
- Colors and layout of tables [114] (with pictures).

See also properties:

- CellBorderColor [534];
- BorderLightColor [532];
- BorderColor [532].

See also properties of table cell:

- BorderColor [569];
- BorderLightColor [570].

1.5.1.1.13 TRVTableItemInfo.CellBorderStyle

Style of border around the table cells

```
property BorderStyle: TRVTableBorderStyle [645];
```

To hide cell borders, set CellBorderWidth [535] = 0.

Default value:

rvtbLowered

See also:

- Undo of assignments to table propeties;
- Colors and layout of tables [114] (with pictures).

See also properties:

- BorderStyle [533];
- CellBorderWidth [535];
- CellBorderColor [534];
- CellBorderLightColor [534].

1.5.1.1.14 TRVTableItemInfo.CellBorderWidth

Width of border around the cells of table, pixels

```
property CellBorderWidth: TRVStyleLength [644];
```

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units [404] of the linked TRVStyle [385] component.

Set to 0 to hide border.

If value of this property <> BorderWidth [534], or if it <> 1 pixel, it can be saved to HTML only with use of CSS

Default value:

0

See also:

- Undo of assignments to table propeties;
- Colors and layout of tables [114] (with pictures).

See also properties:

- BorderWidth [534];
- CellBorderStyle [535];
- CellBorderColor [534];
- CellBorderLightColor [534].

1.5.1.1.15 TRVTableItemInfo.CellHPadding

Horizontal distance from the cell content to the cell border.

```
property CellHPadding: TRVStyleLength[644];
```

(introduced in v10)

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units[404] of the linked TRVStyle[385] component.

This value affects all cells in the table.

If the cell is rotated[573], CellHPadding and CellVPadding[537] are not switched.

Default value:

1

Note: this default value assumes that RVStyle.Units[404]=*rvstuPixels*, otherwise it could be too small.

See also:

- Undo of assignments to table propeties;
- Colors and layout of tables[114] (with pictures).

See also properties:

- CellVPadding[537];
- CellPadding[536].

1.5.1.1.16 TRVTableItemInfo.CellHSpacing

Horizontal spacing between cells.

```
property CellHSpacing: TRVStyleLength[644];
```

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units[404] of the linked TRVStyle[385] component.

This value can be negative. For example, you can assign CellHSpacing=-CellBorderWidth[535] to collapse cell borders horizontally.

Default value:

2

Note: this default value assumes that RVStyle.Units[404]=*rvstuPixels*, otherwise it could be too small.

See also:

- Undo of assignments to table propeties;
- Colors and layout of tables[114] (with pictures).

See also properties:

- CellVSpacing[537];
- BorderHSpacing[532];
- BorderVSpacing[533].

1.5.1.1.17 TRVTableItemInfo.CellPadding

Distance between cell content and cell border.

```
property CellPadding: TRVStyleLength[644];
```

This property is maintained for backward compatibility. Assigning to this property assigns values to CellHPadding[536] and CellVPadding[537]. Reading this property returns Round((CellHPadding[536]+CellVPadding[537])/2).

1.5.1.1.18 TRVTableItemInfo.Cells

Provides indexed access to all cells in the table.

```
property Cells[Row, Col: Integer]: TRVTableCellData[567];
```

This is the key property of table.

Row can be in range from 0 to RowCount[543]-1. **Col** can be in range from 0 to ColCount[538]-1.

Each row has the same number of cells.

Attention: the Borland's grid uses Cells[Col,Row] instead.

Some cells can be equal to *nil* due to cell-merging.

See also properties:

- Rows[543].

See also:

- Table Cells Overview[113].

1.5.1.1.19 TRVTableItemInfo.CellVPadding

Vertical distance from the cell content to the cell border.

```
property CellVPadding: TRVStyleLength[644];
```

(introduced in v10)

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units[404] of the linked TRVStyle[385] component.

This value affects all cells in the table.

If the cell is rotated[573], CellHPadding[536] and CellVPadding are not switched.

Default value:

1

Note: this default value assumes that RVStyle.Units[404]=*rvstuPixels*, otherwise it could be too small.

See also:

- Undo of assignments to table propeties;
- Colors and layout of tables[114] (with pictures).

See also properties:

- CellHPadding[536];
- CellPadding[536].

1.5.1.1.20 TRVTableItemInfo.CellVSpacing

Vertical spacing between cells.

```
property CellVSpacing: TRVStyleLength[644];
```

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units[404] of the linked TRVStyle[385] component.

This value can be negative. For example, you can assign CellVSpacing=-CellBorderWidth[535] to collapse cell borders vertically.

Default value:

2

Note: this default value assumes that RVStyle.Units[404]=*rvstuPixels*, otherwise it could be too small.

See also:

- Undo of assignments to table propeties;
- Colors and layout of tables [114] (with pictures).

See also properties:

- CellHSpacing [536];
- BorderHSpacing [532];
- BorderVSpacing [533].

1.5.1.1.21 TRVTableItemInfo.ColBandSize, Row BandSize

The properties specify the number of rows/columns in alternating row bands

```
property RowBandSize: Integer;
property ColBandSize: Integer;
```

(introduced in version 17)

RowBandSize specify the count of rows in row bands.

ColBandSize specify the count of columns in column bands.

Bands are used to implement alternate colors, see the topic about color properties of rows and columns [546].

The heading (or the first row) and the first column may be excluded from bands.

Default values

1

1.5.1.1.22 TRVTableItemInfo.ColCount

Returns the number of columns in the table.

```
property ColCount: Integer
```

(introduced in v10)

If RowCount [543]>0, this property returns Rows [543][0].Count (all rows have the same number of cells). If RowCount [543]=0, this property returns 0.

See also properties:

- Cells [537];
- Rows [543].

1.5.1.1.23 TRVTableItemInfo.Color

Background color of the table

```
property Color: TColor;
```

Set to *clNone* for transparent background. You can specify the background Opacity [540].

If the table has a background image [530], this image is drawn on top of color filling.

Default value:

clWindow

See also:

- Undo of assignments to table properties;
- Colors and layout of tables [114] (with pictures).

1.5.1.1.24 TRVTableItemInfo.HeadingRowCount

Specifies the number of rows that will be repeated on each printer page with this table.

```
property HeadingRowCount: Integer;
```

(introduced in version 1.7)

If these rows overlap next rows (due to cell merging), this property is ignored on pagination.

A special color (HeadingRowColor[546]) can be applied to heading rows (even if they overlap next rows). If **HeadingRowCount**=0, this color is applied to the first row.

Default value:

0

See also:

- PrintOptions[542],
- ColBandSize, RowBandSize[538]

1.5.1.1.25 TRVTableItemInfo.HOutermostRule

Specifies if the horizontal rules should be drawn between the table border and the outermost cells.

```
property HOutermostRule: Boolean;
```

These rules can be visible only of other rules are visible (HRuleWidth[540]>0)

Note about rules.

Default value:

False

See also:

- Undo of assignments to table properties;
- Table rules[115] (with pictures).

See also properties:

- VOutermostRule[545];
- HRuleWidth[540];
- HRuleColor[539].

1.5.1.1.26 TRVTableItemInfo.HRuleColor

Specifies the color of horizontal rules (horizontal lines between cells)

```
property HRuleColor: TColor;
```

Horizontal rules are visible only if HRuleWidth[540]>0

Note about rules.

Default value:

clWindowText

See also:

- Undo of assignments to table properties;
- Table rules[115] (with pictures).

See also properties:

- VRuleColor[545];
- HRuleWidth[540];

- HOutermostRule[539].

1.5.1.1.27 TRVTableItemInfo.HRuleWidth

Specifies the width of horizontal rules (horizontal lines between cells)

```
property HRuleWidth: TRVStyleLength[644];
```

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units[404] of the linked TRVStyle[385] component.

Set to 0 to hide horizontal rules (hidden by default).

Note about rules.

Default value:

0

See also

- Undo of assignments to table properties;
- Table rules[115] (with pictures).

See also properties

- VRuleWidth[545];
- HRuleColor[539];
- HOutermostRule[539].

1.5.1.1.28 TRVTableItemInfo.Opacity

Opacity of table background.

```
property Opacity: TRVOpacity[637];
```

(introduced in version 16)

This value is used if Color[538]<>*clNone*. The opacity is applied only to background color filling, not to background image.

Default value:

100000 (i.e. 100%)

See also:

- Colors and layout of tables[114].

1.5.1.1.29 TRVTableItemInfo.Options

Specifies options for the table.

```
type
  TRVTableOption = (rvtoEditing,
    rvtoRowSizing, rvtoColSizing,
    rvtoRowSelect, rvtoColSelect,
    rvtoNoCellSelect,
    rvtoRTFSaveCellPixelBestWidth,
    rvtoHideGridLines,
    rvtoCellBelowBorders,
    rvtoOverlappingCorners,
    rvtoRTFAllowAutofit,
    rvtoIgnoreContentWidth,
    rvtoIgnoreContentHeight);
  TRVTableOptions = set of TRVTableOption;
```

```
property Options: TRVTableOptions;
```

Default value

[*rvtoEditing, rvtoRowSizing, rvtoColSizing, rvtoRowSelect, rvtoColSelect*]

Options for Editing

Option	Meaning
rvtoEditing	If set (default), users can edit content of cells using inplace editor. (table must be inserted in TRichViewEdit 282 or TDBRichViewEdit 377)

Options for Resizing and Selection

Option	Meaning
rvtoRowSizing *rvtoColSizing*	If set (default), users can resize individual rows/columns with the mouse. (table must be inserted in TRichViewEdit 282 or TDBRichViewEdit 377)
rvtoRowSelect, *rvtoColSelect*	If set (default), user can select rows/columns by clicking on the left/top border of the table (when cursor has a special vertical/ horizontal arrow shape) The options have no effect if the table is inserted in RichView with *rvoAllowSelection* excluded from Options 146.
rvtoNoCellSelect	If set, multicell selection by mouse is not allowed (remove also *rvtoRowSelect* and *rvtoColSelect* to disallow making multicell selection completely.

Layout Options

Option	Meaning
rvtoIgnoreContentWidth	If set, contents of cells are completely ignored when calculating widths of table columns.Wide pictures, controls, etc. are clipped.
rvtoIgnoreContentHeight	If set, heights of rows are calculated basing only on BestHeight 568 properties of cells. But (unlike *rvtoIgnoreContentWidth*), heights of cells having BestHeight=0 are still calculated basing on their content.

Visual Options

Option	Meaning
rvtoHideGridLines	If not set (default), invisible borders are shown using GridColor, GridReadOnlyColor, GridStyle, GridReadOnlyStyle 392 properties of the linked TRVStyle. It's not recommended to use this option. Use rvoShowGridLines

	option in TRichView.Options[146] instead.
rvtoCellBelowBorders	If not set (default), space below invisible side[575] of border has color[538] of tables. If set, color of cell[570] is used. 
rvtoOverlappingCorners	If not set (default), colored borders (CellBorderStyle[535] =*rvtbColor*) have three-dimensional appearance. If not set, they are flat. The effect can be noticed for wide borders with some invisible sides[575]. 

RTF and DocX Export Options

Option	Meaning
rvtoRTFAllowAutofit	If set, tables with default width (BestWidth[531]=0) are exported to RTF and DocX as fit-by-content tables. Off by default. See also: RichViewTableDefaultRTFAutofit[657]
rvtoRTFSaveCellPixelBestWidth	If set, exact sizes of cells are saved to RTF and DocX. Off by default.

The options above are explained in the topic about table RTF and DocX export[123].

1.5.1.1.30 TRVTableItemInfo.PrintOptions

Specifies options for table printing.

```
type
  TRVTablePrintOption =
    (rvtoHalftoneBorders, rvtoRowsSplit,
     rvtoWhiteBackground, rvtoContinue);
  TRVTablePrintOptions = set of TRVTablePrintOption;

property PrintOptions: TRVTablePrintOptions;
```

Option	Meaning
rvtoHalftoneBorders	If set (default), table and cells borders will be printed in dithered colors, but corners will be rounded.

	If cleared, borders will be printed in solid colors (usually black or white), but corners will be strictly defined, like on screen. Affects to 3d borders (see BorderStyle 533 and CellBorderStyle 535) and borders with some invisible lines. Forced compromise.
rvtoRowsSplit	Clear this option to forbid printing this table on several pages.
rvtoWhiteBackground	If set, colors of table and cells backgound will be ignored and printed in white. See also TRVPrint.TransparentBackground 517
rvtoContinue	If set, and the table has heading rows 539, heading rows are displayed only from the second table page

Default value:

[*rvtoHalftoneBorders,rvtoRowsSplit*]

See also:

- HeadingRowCount 539.

1.5.1.1.31 TRVTableItemInfo.RowCount

Returns the number of rows in the table.

```
property RowCount: Integer
```

(introduced in v10)

See also properties:

- ColCount 538;
- Cells 537;
- Rows 543.

1.5.1.1.32 TRVTableItemInfo.Rows

Provides indexed access to all rows in the table.

```
property Rows: TRVTableRows 578;
```

Rows is a list of table rows (TRVTableRow 577), each row is a list of cells (TRVTableCellData 567). Each row has the same number of cells.

The specific cell can be accessed as Rows[<Row>][<Col>] as well as Cells 537[<Row>,<Col>].

Use Rows to access properties of rows, for example Rows[<Row>].VAlign 578.Otherwise, Cells 537 is more convenient.

See also:

- Table Cells Overview 113

1.5.1.1.33 TRVTableItemInfo.TextColSeparator

Value of this property is used when saving table in text file. This string will be written between cells.

```
property TextColSeparator: String;
```

When saving to Unicode text files, RichView writes Unicode representation of this string.

Default value

#13#10 (CR+LF)

You can set TextColSeparator to TAB, if you wish, but do not forget that there can be CR+LFs inside cells.

See also properties:

- TextRowSeparator 544.

See also methods of TRichView:

- SaveText 221;
- GetSelText 204.

See also:

- Export of Tables 123.

1.5.1.1.34 TRVTableItemInfo.TextRow Separator

Value of this property is used when saving table in text file. This string will be written between rows.

```
property TextCowSeparator: String;
```

When saving to Unicode text files, RichView writes Unicode representation of this string.

Default value:

#13#10 (CR+LF)

See also properties:

- TextColSeparator 543.

See also methods of TRichView:

- SaveText 221;
- GetSelText 204.

See also:

- Export of Tables 123.

1.5.1.1.35 TRVTableItemInfo.VisibleBorders

Allows to hide some sides of table border.

```
property VisibleBorders: TRVBooleanRect 611;
```

(introduced in v10)

Direct assignment to this property cannot be undone/redone by user (use SetTableVisibleBorders 563)

Default value:

True, *True*, *True*, *True* (all sides are visible)

See also properties:

- BorderWidth 534;
- BorderColor 532;
- BorderLightColor 532;
- BorderStyle 533.

See also properties of table cell:

- VisibleBorders 575.

1.5.1.1.36 TRVTableItemInfo.VOutermostRule

Specifies if the vertical rules should be drawn between the table border and the outermost cells.

```
property VOutermostRule: Boolean;
```

These rules can be visible only of other rules are visible (VRuleWidth[545]>0)

Note about rules.

Default value:

False

See also:

- Undo of assignments to table properties;
- Table rules[115] (with pictures).

See also properties:

- HOutermostRule[539];
- VRuleWidth[545];
- VRuleColor[545].

1.5.1.1.37 TRVTableItemInfo.VRuleColor

Specifies the color of vertical rules (vertical lines between cells)

```
property VRuleColor: TColor;
```

Vertical rules are visible only if VRuleWidth[545]>0

Note about rules.

Default value:

clWindowText

See also:

- Undo of assignments to table properties;
- Table rules[115] (with pictures).

See also properties:

- HRuleColor[539];
- VRuleWidth[545];
- VOutermostRule[545].

1.5.1.1.38 TRVTableItemInfo.VRuleWidth

Specifies the width of vertical rules (vertical lines between cells)

```
property VRuleWidth: TRVStyleLength[644];
```

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units[404] of the linked TRVStyle[385] component.

Set to 0 to hide vertical rules (hidden by default).

Note about rules.

Default value:

0

See also:

- Undo of assignments to table properties;

- Table rules[115] (with pictures).

See also properties:

- HRuleWidth[540];
- VRuleColor[545];
- VOutermostRule[545].

1.5.1.1.39 TRVTableItemInfo's row and column colors

A set of properties defining default colors of cells in specific rows and columns.

```
property HeadingRowColor: TColor;
property LastRowColor: TColor;
property OddRowsColor: TColor;
property EvenRowsColor: TColor;
property FirstColumnColor: TColor;
property LastColumnColor: TColor;
property OddColumnsColor: TColor;
property EvenColumnsColor: TColor;
```

(introduced in version 17)

By default, all cells are transparent, and you can see the table background through cells (see Color[538] property).

Using the properties listed in this topic, you can define default colors of cells in the corresponding rows and columns.

Finally, you can define the color of the specified cell in Cell.Color[570] property.

Property	Meaning
HeadingRowColor	Color of heading rows[539]. If this table has no heading rows, but **HeadingRowColor**<>*clNone*, this color is used for the first row.
LastRowColor	Color of the last row
OddRowsColor	Color of odd row bands
EvenRowsColor	Color of even row bands
FirstColumnColor	Color of the first column
LastColumnColor	Color of the last column
OddColumnsColor	Color of odd row bands
EvenColumnsColor	Color of even row bands

if **FirstColumnColor**<>*clNone*, column bands are started from the second column, otherwise from the first column. Each band has ColBandSize[538] columns.

Row bands are started from the first column after the table heading rows. If HeadingRowCount[539]>0, it defines the number of heading rows. Otherwise, if HeadingRowColor<>clNone, one heading row is assumed. Otherwise, row bands start from the first row. Each band has RowBandSize[538] rows.

Colors have the following priorities (in descending order) : HeadingRowColor, LastRowColor, FirstColumnColor, LastColumnColor, OddColumnsColor and EvenColumnsColor, OddRowsColor and EvenRowsColor. Only properties having values different from *clNone* are taken into account.

Default values

clNone

See also

- Color 538
- HeadingRowCount 539
- ColBandSize, RowBandSize 538
- Cell.Color 570
- Cell.Opacity 571

1.5.1.2 Methods

1.5.1.2.1 TRVTableItemInfo.AssignProperties

Assigns all table properties from the **Source**.

```
procedure AssignProperties(Source: TRVTableItemInfo 528);
```

(introduced in v1.9)

Cells and their contents are not assigned.

1.5.1.2.2 TRVTableItemInfo.CanMergeCells

Returns: is cell-merging of the specified cells possible?

```
function CanMergeCells(TopRow, LeftCol, ColSpan, RowSpan: Integer;
  AllowMergeRC: Boolean): Boolean;
```

Potential area for cell-merging is specified in **TopRow**, **TopCol**, **ColSpan**, **RowSpan**.

TopRow, **TopCol** – position of the top left cell for cell-merging.

ColSpan – number of columns for merging.

RowSpan – number of rows for merging.

If merging will be performed, the resulting cell will be Cells 537[**TopRow**, **TopCol**] with ColSpan 570=**ColSpan** and RowSpan 574=**RowSpan**.

Merging can be done, if

1. Cells 537[**TopRow**, **TopCol**]<>*nil*.
2. This area defines a rectangular piece of table (there are no cells overlapping its border due to previous cell-merging). This requirement covers the 1st one.
3. Either Cells 537[**TopRow**, **TopCol**].ColSpan 570 < **ColSpan** or Cells 537[**TopRow**, **TopCol**].RowSpan 574 < **RowSpan**.
4. If **AllowMergeRC**=*True*, this function returns *False*, if one or more rows or columns will contain only *nil*s after merging. It's not recommended to have such rows/columns, so you can either forbid such operation or (better) delete them with DeleteEmptyRows 550 and DeleteEmptyCols 550.

See also:

- CanMergeSelectedCells 548;
- MergeCells 556;
- MergeSelectedCells 557.

See also properties:

- ColCount 538;
- RowCount 543.

1.5.1.2.3 TRVTableItemInfo.CanMergeSelectedCells

Returns: is cell-merging of the selected cells possible?

```
function CanMergeSelectedCells(AllowMergeRC: Boolean): Boolean;
```

The same as CanMergeCells [547], but potential area for cell-merging is the selection in the table.

If **AllowMergeRC**=*True*, this function returns *False*, if one or more rows or columns will contain only *nil*s after merging. It's not recommended to have such rows/columns, so you can either forbid such operation or (better) delete them with DeleteEmptyRows [550] and DeleteEmptyCols [550].

This method can be used to determine if the selection in the table has a rectangular shape.

This method must be called when the document is formatted.

See also:

- CanMergeCells [547];
- MergeSelectedCells [557];
- MergeCells [556].

1.5.1.2.4 TRVTableItemInfo.Changed

"Informs" table about updates in its content.

```
procedure Changed;
```

After performing operations on table which was already inserted in document you should reformat this document.

It is usually done by BeginItemModify [301]/EndItemModify [307] (if the table was inserted in TRichViewEdit [282]) or Format [182] (if the table was inserted in TRichView [128]).

But you can realize that in some cases table will not be updated even after formatting. This happens because the table does not "know" about changes in it, so it reuses the previous formatting.

The most of operations on table (including all methods of TRVTableItemInfo, and editing operations in cell inplace-editor) "inform" table about changes in its content and thus allowing subsequent reformatting. But viewer-style operations performed on cells do not "inform" table, and their result cannot be shown after reformatting. You can inform table about changes in its content by this method.

Example

```
table := TRVTableItemInfo(MyRichView.GetItem[190](ItemNo));
table.Cells[0,0].Clear[175];
table.Cells[0,0].AddNL[576]('One more line',0,0);
table.Changed;
MyRichView.Format[182];
```

Note: since version 1.7, it's not required to call table.Changed before Format [182]. But it is still useful when using BeginItemModify [301]/EndItemModify [307].

1.5.1.2.5 TRVTableItemInfo.Create

Creates a new TRVTableItemInfo object

```
constructor Create(RVData: TPersistent); override;
```

Parameter:

RVData – RVData [151] property of TRichView [128] or TRichViewEdit [282], where you wish to insert this table.

This constructor creates a new table with one row and one column.

There is more convenient constructor, CreateEx [549].

See also methods of TCustomRichView:

- AddItem 166.

See also methods of TCustomRichViewEdit:

- InsertItem 320.

1.5.1.2.6 TRVTableItemInfo.CreateEx

Creates a new TRVTableItemInfo object having **nRows** rows and **nCols** columns.

```
constructor CreateEx(nRows, nCols: Integer; AMainRVData: TCustomRVData);
```

Parameters:

nRows – count of rows.

nCols – count of columns.

RVData – RVData 151 property of TRichView 128 or TRichViewEdit 282, where you wish to insert this table.

If **nRows**=0, then **nCols** is ignored, and 0x0 table is created. A number of columns is stored only if table.RowCount 543>0.

Example

```
var table:TRVTableItemInfo;
...
table := TRVTableItemInfo.CreateEx(5, 5, MyRichViewEdit.RVData 151);
table.Color 538 := clYellow;
table.Cells 537[0,0].AddNL 576('Hello',0,0);
MyRichViewEdit.InsertItem 320('5x5 table', table);
```

See also methods:

- Create 548.

See also methods of TCustomRichView:

- AddItem 166.

See also methods of TCustomRichViewEdit:

- InsertItem 320.

1.5.1.2.7 TRVTableItemInfo.DeleteCols

Deletes **Count** columns starting from the **Index**-th column.

```
procedure DeleteCols(Index, Count: Integer; DecreaseWidth: Boolean);
```

Parameters:

Index – index of the first column to delete, in range 0..ColCount 538-1.

Count – number of columns to delete.

if **DecreaseWidth** = *True*, BestWidth 569s of cells overlapping the deleted columns (due to cell-merging) will be decreased (recommended).

See also:

- Undo note.

See also methods:

- DeleteSelectedCols 551;
- DeleteRows 550;
- DeleteEmptyCols 550.

1.5.1.2.8 TRVTableItemInfo.DeleteEmptyCols

Deletes all columns containing only *nil*-cells.

```
procedure DeleteEmptyCols;
```

Nil-cells occur due to cell-meging.

All-*nil* columns occur after cell-merging or deleting rows.

This method calls DeleteCols[549] with *DecreaseWidth* = *False*.

See also:

- Undo note.

See also methods:

- DeleteEmptyRows[550];
- DeleteSelectedCols[551];
- DeleteCols[549].

1.5.1.2.9 TRVTableItemInfo.DeleteEmptyRows

Deletes all rows containing only *nil*-cells.

```
procedure DeleteEmptyRows;
```

Nil-cells occur due to cell-meging.

All-nil rows occur after cell-merging or deleting columns.

This method calls DeleteRows[550] with DecreaseHeight = *False*.

See also:

- Undo note.

See also:

- DeleteEmptyCols[550];
- DeleteSelectedRows[551];
- DeleteRows[550].

1.5.1.2.10 TRVTableItemInfo.DeleteRows

Deletes **Count** rows starting from the **Index**-th row.

```
procedure DeleteRows(Index, Count: Integer; DecreaseHeight: Boolean);
```

Parameters:

Index – index of the first row to delete, in range 0..RowCount[543]-1.

Count – number of rows to delete.

if **DecreaseHeight** = *True*, BestHeight[568]s of cells overlapping the deleted rows (due to cell-merging) will be decreased (recommended).

See also:

- Undo note.

See also methods:

- DeleteSelectedRows[551];
- DeleteCols[549];
- DeleteEmptyRows[550].

1.5.1.2.11 TRVTableItemInfo.DeleteSelectedCols

Deletes all columns containing selected cells

```
procedure DeleteSelectedCols;
```

This method deletes all columns containing the selected cells, even if not all cells in these columns are selected.

The cell that is being edited is considered selected.

This method must be called when the document is formatted.

See also:

- Undo note.

See also methods:

- DeleteSelectedRows 551;
- DeleteCols 549;
- DeleteEmptyCols 550.

1.5.1.2.12 TRVTableItemInfo.DeleteSelectedRows

Deletes all rows containing selected cells.

```
procedure DeleteSelectedRows;
```

This method deletes all rows containing selected cells, even if not all cells in these rows are selected.

The cell that is being edited is considered selected.

This method must be called when the document is formatted.

See also:

- Undo note.

See also methods:

- DeleteSelectedCols 551;
- DeleteCols 549;
- DeleteEmptyCols 550.

1.5.1.2.13 TRVTableItemInfo.Deselect

Removes selection from the table.

```
procedure Deselect;
```

All selected cells become not selected.

See also:

- Overview of selection in tables 116.

See also methods:

- Select 558;
- SelectRows 558;
- SelectCols 558.

1.5.1.2.14 TRVTableItemInfo.Destroy

Destroys an instance of TRVTableItemInfo.

```
destructor Destroy; override;
```

Do not call Destroy directly in an application. Tables are freed by the control where they were inserted.

1.5.1.2.15 TRVTableItemInfo.EditCell

Activates editing of the specified cell..

```
procedure EditCell(Row,Col: Integer);
```

Parameters:

Row and **Col** defines the position of cell to edit. **Row** must be in range 0..RowCount [543]-1, **Col** must be in range 0..ColCount [538]-1.

Cells [552][**Row**, **Col**] must not be *nil*.

This method creates inplace-editor for this cell.

If this table is nested in another table, this method also initializes inplace-editor in the parent table.

This is an equivalent for Cells [552][Row, Col].Edit [610].

This method must be called when the document is formatted.

See also:

- Overview of cell editing [117].

See also methods:

- GetNormalizedSelectionBounds [553];
- GetEditedCell [552].

1.5.1.2.16 TRVTableItemInfo.GetCellAt

Returns cell at the specified coordinates.

```
function GetCellAt(X,Y: Integer; var Row,Col: Integer): Boolean;
```

Input parameters:

X, **Y** – coordinates, relative to the top left corner of the table.

Output parameters:

Row, **Col** – receive the position of cell.

This method must be called when the document is formatted.

See also methods of TCustomRichView:

- GetItemCoords [191].

1.5.1.2.17 TRVTableItemInfo.GetEditedCell

Returns inplace editor for the currently edited cell, or *nil* if there is no such cell.

```
function GetEditedCell(var Row,Col: Integer): TCustomRichViewEdit [282];
```

Output parameters:

Row and **Col** receive position of the cell that is being edited.

This method must be called when the document is formatted.

See also:

- Overview of selection in tables [116];
- Overview of cell editing [117].

See also methods:

- GetNormalizedSelectionBounds [553];
- EditCell [552].

1.5.1.2.18 TRVTableItemInfo.GetNormalizedSelectionBounds

Returns selection in table in convenient form.

```
function GetNormalizedSelectionBounds(IncludeEditedCell: Boolean;
         var TopRow, LeftCol, ColSpan, RowSpan: Integer): Boolean;
```

Input parameter:

If *IncludeEditedCell* = *True*, this function returns position of the currently edited cell (if there is no selection and some cell is currently in editing state).

Output parameters:

TopRow, **TopCol** – receive the top left corner of the selection (warning: selection is not always rectangular!)

ColSpan, **RowSpan** – how many columns and rows the selection includes. Both values are >= 1.

Warning: it will be wrong to iterate through selected cells as:

```
for r := TopRow to  TopRow+RowSpan-1 do
  for c := LeftCol to TopCol+ColSpan-1 do
    if table.Cells[537][r,c]<>nil then ...
```

This cycle can miss some selected cells, because there can be some cells to the left of **LeftCol** and to the top of **TopRow**, which are also in selection because of spanning.

The correct example is in the topic about IsCellSelected[556].

Some operations are possible only on rectangular selection (for example, cell merging). It can be checked using CanMergeSelectedCells[548].

This method must be called when the document is formatted.

Return value:

True, if selection exists, *False* otherwise.

See also:

- Overview of selection in tables[116]

See also methods:

- GetSelectionBounds[553];
- GetEditedCell[552].

1.5.1.2.19 TRVTableItemInfo.GetSelectionBounds

Returns selection in table.

```
function GetSelectionBounds(var StartRow, StartCol, RowOffs, ColOffs: Integer): Boolean;
```

Output parameters:

Cells[537][**StartRow**, **StartCol**] is a cell where selection is started. Selection always includes this cell.

if **RowOffs**>0, the selection includes **RowOffs** rows below. If **RowOffs**<0, the selection includes abs(**RowOffs**) rows above.

The same is for **ColOffs**.

Selection includes all cells intersected with imaginary rectangle that fully includes Cells[**StartRow**, **StartCol**] and Cells[**StartRow**+**RowOffs**, **StartCol**+**ColOffs**]. Warning: these cells can be *nil* because of merging.

For example, the returned value (3,3,0,0) means that only one cell is selected: Cells[3,3].

Selection can have a rectangular shape or not (because of merging). Some operations are possible only on rectangular selection (for example, cell merging). It can be checked using CanMergeSelectedCells[548].

This method returns selection as it is stored internally. For practical use, GetNormalizedSelectionBounds[553] is more convenient.

This method must be called when the document is formatted.

Return value:

True, if selection exists, *False* otherwise.

See also:

- Overview of selection in tables 116.

See also methods:

- GetNormalizedSelectionBounds 553;
- Select 558;
- Deselect 551.

1.5.1.2.20 TRVTableItemInfo.InsertCols

Inserts **Count** columns in the position specified by **Index**.

```
procedure InsertCols(Index, Count, CopyIndex: Integer);
```

Parameters:

Index – index of the column to insert before (or ColCount 538 to insert after the last column).

Count – number of columns to insert.

CopyIndex – if <>-1, this is an index (before the insertion) of column to copy attributes for new columns from. Colors, background, border, size, text and paragraph styles are copied.

See also:

- Undo note.

See also methods:

- InsertRows 555;
- InsertColsLeft 554, InsertColsRight 554;
- DeleteCols 549.

1.5.1.2.21 TRVTableItemInfo.InsertColsLeft

Inserts **Count** columns to the right of the selected cells (or the cell being edited)

```
procedure InsertColsLeft(Count: Integer);
```

Does nothing if the table has no selection or editing cell.

This method must be called when the document is formatted.

See also:

- Undo note.

See also methods:

- InsertColsRight 554;
- InsertCols 554;
- InsertRowsAbove 555, InsertRowsBelow 555;
- DeleteSelectedCols 551.

1.5.1.2.22 TRVTableItemInfo.InsertColsRight

Inserts **Count** columns to the right of the selected cells (or the cell being edited)

```
procedure InsertColsRight(Count: Integer);
```

Does nothing if the table has no selection or editing cell.

This method must be called when the document is formatted.

See also:

- Undo note.

See also methods:

- InsertColsLeft 554;
- InsertCols 554;
- InsertRowsAbove 555, InsertRowsBelow 555;
- DeleteSelectedCols 551.

1.5.1.2.23 TRVTableItemInfo.InsertRows

Inserts **Count** rows in the position specified by **Index**.

```
procedure InsertRows(Index, Count, CopyIndex: Integer);
```

Parameters:

Index – index of the row to insert before (or RowCount 543 to insert after the last row).

Count – number of rows to insert.

CopyIndex – if <>-1, this is an index (before the insertion) of row to copy attributes for new rows from. Colors, background, border, size, text and paragraph styles are copied.

See also:

- Undo note.

See also methods:

- InsertCols 554;
- InsertRowsAbove 555, InsertRowsBelow 555;
- DeleteRows 550.

1.5.1.2.24 TRVTableItemInfo.InsertRowsAbove

Inserts **Count** rows above the selected cells (or the cell being edited)

```
procedure InsertRowsAbove(Count: Integer);
```

Does nothing if the table has no selection or editing cell.

This method must be called when the document is formatted.

See also:

- Undo note.

See also methods:

- InsertRowsBelow 555;
- InsertRows 555;
- InsertColsLeft 554, InsertColsRight 554;
- DeleteSelectedRows 551.

1.5.1.2.25 TRVTableItemInfo.InsertRowsBelow

Inserts **Count** rows below the selected cells (or the cell being edited)

```
procedure InsertRowsBelow(Count: Integer);
```

Does nothing if the table has no selection or editing cell.

This method must be called when the document is formatted.

See also:

- Undo note.

See also methods:

- InsertRowsAbove 555;
- InsertRows 555;

- InsertColsLeft[554], InsertColsRight[554];
- DeleteSelectedRows[551].

1.5.1.2.26 TRVTableItemInfo.IsCellSelected

Returns: is the specified cell selected or not.

```
function IsCellSelected(Row, Col: Integer): Boolean;
```

Example: iteration through all selected cells

```
for r := 0 to table.RowCount[543]-1 do
  for c := 0 to table.ColCount[538]-1 do
    if (table.Cells[537][r,c]<>nil) and
       table.IsCellSelected(r,c) then ...
```

This method must be called when the document is formatted.

See also:

- Overview of selection in tables[116].

See also methods:

- GetNormalizedSelectionBounds[553].

1.5.1.2.27 TRVTableItemInfo.LoadFromStream

Loads table from **Stream**.

```
procedure LoadFromStream(Stream: TStream);
```

Table must be saved using SaveToStream[557] or SaveRowsToStream[557] methods. Table can be loaded correctly only if collections of text, paragraph and list styles in TRVStyle[385] are the same as at the moment of saving.

Applications rarely, if ever, call LoadFromStream directly. Usually tables are loaded from RVF files or streams[59] together with main documents.

This method must be used in conjunction with BeforeLoading and AfterLoading. AfterLoading must be called when the table is inserted in the document.

```
Table.BeforeLoading(rvlfRVF);
Table.LoadFromStream(Stream);
if RichViewEdit1.InsertItem[320]('', Table) then
  Table.AfterLoading(rvlfRVF);
```

1.5.1.2.28 TRVTableItemInfo.MergeCells

Merges the specified cells in one.

```
procedure MergeCells(TopRow, LeftCol, ColSpan, RowSpan: Integer; AllowMergeRC: Boolean);
```

Parameters:

TopRow, **LeftCol** – position of the top left cell for cell-merging.

ColSpan – number of columns for merging.

RowSpan – number of rows for merging.

The resulting cell is Cells[537][**TopRow**, **TopCol**] having ColSpan[570]=**ColSpan** and RowSpan[574]=**RowSpan**.

This method fails (does nothing) if the parameters do not define a rectangle (because of previous cell merging). You can check the possibility of merging using CanMergeCells[547].

AllowMergeRC = *False* forbids to perform merging, if it will cause appearing rows or columns containing only *nils*. It's not recommended to have such rows/columns, so you can either forbid such operation or to delete them using DeleteEmptyRows[550] and DeleteEmptyCols[550] (the latter is recommended).

See also:

- Undo note.

See also methods:

- CanMergeCells [547];
- MergeSelectedCells [557];
- GetNormalizedSelectionBounds [553];
- UnmergeCells [564].

1.5.1.2.29 TRVTableItemInfo.MergeSelectedCells

Merges the selected cells.

```
procedure MergeSelectedCells(AllowMergeRC: Boolean);
```

This method fails (does nothing) if the selection does not have a rectangular shape (because of previous cell merging). You can check possibility of merging using CanMergeSelectedCells [548].

AllowMergeRC = *False* forbids to perform merging, if it will cause appearing rows or columns containing only *nil*s. It's not recommended to have such rows/columns, so you can either forbid such operation or delete them with DeleteEmptyRows [550] and DeleteEmptyCols [550] (the latter is recommended).

This method must be called when the document is formatted.

See also:

- Undo note.

See also methods:

- CanMergeSelectedCells [548];
- MergeCells [556];
- GetNormalizedSelectionBounds [553];
- UnmergeSelectedCells [564].

1.5.1.2.30 TRVTableItemInfo.SaveRowsToStream

Saves the specified rows of table (from **Index** to **Index**+**Count**-1) to **Stream**.

```
procedure SaveRowsToStream(Stream: TStream; Index, Count: Integer);
```

(introduced in version 10)

The output is compatible with SaveToStream [557].

The result will be correct even if there are cells that overlap these rows because of RowSpan [574]>1.

See also method:

- LoadFromStream [556].

See also:

- Example how to divide table in two parts [126].

1.5.1.2.31 TRVTableItemInfo.SaveToStream

Saves table to **Stream**.

```
procedure SaveToStream(Stream: TStream);
```

Applications rarely, if ever, call SaveToStream directly. Usually tables are saved to RVF files or streams [59] with main documents.

See also method:

- LoadFromStream [556];
- SaveRowsToStream [557].

1.5.1.2.32 TRVTableItemInfo.Select

Selects the range of cells in the table

```
procedure Select(StartRow, StartCol, RowOffs, ColOffs: Integer);
```

This method selects range (rectangle, if there is no cells-merging) of cells, starting from the cell (**StartRow**, **StartCol**). The parameters have the same meaning as in GetSelectionBounds[553].

Selection includes all cells intersected with imaginary rectangle that fully includes Cells[537][**StartRow**, **StartCol**] and Cells[537][**StartRow**+**RowOffs**, **StartCol**+**ColOffs**].

You cannot remove selection with this method, because Select(StartRow, StartCol, 0,0) selects one cell, use Deselect[551].

This method must be called when the document is formatted.

See also:

- Overview of selection in tables[116]

See also methods:

- SelectRows[558], SelectCols[558];
- GetSelectionBounds[553];
- Deselect[551].

1.5.1.2.33 TRVTableItemInfo.SelectCols

Selects **Count** columns starting from **StartCol** (to the right)

```
procedure SelectCols(StartCol, Count: Integer);
```

Parameters:

StartCol – index of the first column to select, in range 0..ColCount[538]-1.

Count – number of columns to select.

This method must be called when the document is formatted.

See also:

- Overview of selection in tables[116]

See also methods:

- SelectRows[558],
- Select[558].

1.5.1.2.34 TRVTableItemInfo.SelectRows

Selects **Count** rows starting from **StartRow** (down)

```
procedure SelectRows(StartRow, Count: Integer);
```

Parameters:

StartRow – index of the first row to select, in range 0..RowCount[543]-1.

Count – number of rows to select.

This method must be called when the document is formatted.

See also:

- Overview of selection in tables[116]

See also methods:

- SelectCols[558];
- Select[558].

1.5.1.2.35 TRVTableItemInfo.SetCellBackgroundImage

Assigns **Value** to Cells[537][**Row**, **Col**].BackgroundImage[567], as an editing operation.

```
procedure SetCellBackgroundImage(Value: TGraphic; Row,Col: Integer);
```

Unlike a direct assignment to Cells[537][**Row**, **Col**].BackgroundImage[567] property, this method can be undone/redone, if it was called for table inserted in TRichViewEdit (after insertion[320]).

Image in **Value** is copied, so you still need to free memory for this it.

Use this method in conjunction with methods for updating editor window, see example[122].

See also:

- Undo/redo in tables[121];
- Undo/redo in TRichViewEdit[54].

1.5.1.2.36 TRVTableItemInfo.SetCellBackgroundImageFileName

Assigns **Value** to Cells[537][**Row**, **Col**].BackgroundImageFileName[568], as an editing operation.

```
procedure SetCellBackgroundImageFileName(const Value: String; Row,Col: Integer);
```

Unlike a direct assignment to Cells[537][**Row**, **Col**].BackgroundImageFileName[568] property, this method can be undone/redone, if it was called for table inserted in TRichViewEdit (after insertion[320]).

Use this method in conjunction with methods for updating editor window, see example[122].

See also:

- Undo/redo in tables[121];
- Undo/redo in TRichViewEdit[54].

1.5.1.2.37 TRVTableItemInfo.SetCellBackgroundStyle

Assigns **Value** to Cells[537][**Row**, **Col**].BackgroundStyle[568], as an editing operation.

```
procedure SetCellBackgroundStyle(Value: TRVItemBackgroundStyle[636]; Row,Col: Integer);
```

Unlike a direct assignment to Cells[537][**Row**, **Col**].BackgroundStyle[568] property, this method can be undone/redone, if it was called for table inserted in TRichViewEdit (after insertion[320]).

Use this method in conjunction with methods for updating editor window, see example[122]

See also:

- Undo/redo in tables[121];
- Undo/redo in TRichViewEdit[54].

1.5.1.2.38 TRVTableItemInfo.SetCellBestHeight

Assigns **Value** to Cells[537][**Row**, **Col**].BestHeight[568], as an editing operation.

```
procedure SetCellBestHeight(Value: TRVStyleLength[644]; Row,Col: Integer);
```

Unlike a direct assignment to Cells[537][**Row**, **Col**].BestHeight[568] property, this method can be undone/redone, if it was called for table inserted in TRichViewEdit (after insertion[320]).

Use this method in conjunction with methods for updating editor window, see example[122]

See also:

- Undo/redo in tables[121];
- Undo/redo in TRichViewEdit[54].

1.5.1.2.39 TRVTableItemInfo.SetCellBestWidth

Assigns **Value** to Cells[537][**Row**, **Col**].BestWidth[569], as an editing operation.

```
procedure SetCellBestWidth(Value: Integer; Row,Col: Integer);
```

Unlike a direct assignment to Cells[537][**Row**, **Col**].BestWidth[569] property, this method can be undone/redone, if it was called for table inserted in TRichViewEdit (after insertion[320]).

Use this method in conjunction with methods for updating editor window, see example[122].

See also:

- Undo/redo in tables[121];
- Undo/redo in TRichViewEdit[54].

1.5.1.2.40 TRVTableItemInfo.SetCellBorderColor

Assigns **Value** to Cells[537][**Row**, **Col**].BorderColor[569], as an editing operation.

```
procedure SetCellBorderColor(Value: TColor; Row,Col: Integer);
```

(introduced in version 1.6)

Unlike a direct assignment to Cells[537][**Row**, **Col**].BorderColor[569] property, this method can be undone/redone, if it was called for table inserted in TRichViewEdit (after insertion[320]).

Use this method in conjunction with methods for updating editor window, see example[122].

See also:

- Undo/redo in tables[121];
- Undo/redo in TRichViewEdit[54].

1.5.1.2.41 TRVTableItemInfo.SetCellBorderLightColor

Assigns **Value** to Cells[537][**Row**, **Col**].BorderLightColor[570], as an editing operation.

```
procedure SetCellBorderLightColor(Value: TColor; Row,Col: Integer);
```

(introduced in version 1.6)

Unlike a direct assignment to Cells[537][**Row**, **Col**].BorderLightColor[570] property, this method can be undone/redone, if it was called for table inserted in TRichViewEdit (after insertion[320]).

Use this method in conjunction with methods for updating editor window, see example[122].

See also:

- Undo/redo in tables[121];
- Undo/redo in TRichViewEdit[54].

1.5.1.2.42 TRVTableItemInfo.SetCellColor

Assigns **Value** to Cells[537][**Row**, **Col**].Color[570], as an editing operation.

```
procedure SetCellColor(Value: TColor; Row,Col: Integer);
```

Unlike a direct assignment to Cells[537][**Row**, **Col**].Color[570] property, this method can be undone/redone, if it was called for the table inserted in TRichViewEdit (after insertion[320]).

Use this method in conjunction with methods for updating editor window, see example[122].

See also:

- Undo/redo in tables[121];
- Undo/redo in TRichViewEdit[54].

1.5.1.2.43 TRVTableItemInfo.SetCellHint

Assigns **Value** to Cells[537][**Row**, **Col**].Hint[571], as an editing operation.

```
procedure SetCellHint(const Value: String; Row,Col: Integer);
```

Unlike a direct assignment to Cells[537][**Row**, **Col**].Hint[571] property, this method can be undone/redone, if it was called for the table inserted in TRichViewEdit (after insertion[320]).

Use this method in conjunction with methods for updating editor window, see example[122].

See also:

- Undo/redo in tables[121];
- Undo/redo in TRichViewEdit[54].

1.5.1.2.44 TRVTableItemInfo.SetCellOpacity

Assigns **Value** to Cells[537][**Row**, **Col**].Opacity[571], as an editing operation.

```
procedure SetCellOpacity(Value: TRVOpacity[637]; Row,Col: Integer);
```

(introduced in version 16)

Unlike a direct assignment to Cells[537][**Row**, **Col**].Opacity[571] property, this method can be undone/redone, if it was called for the table inserted in TRichViewEdit (after insertion[320]).

Use this method in conjunction with methods for updating editor window, see example[122].

See also:

- Undo/redo in tables[121];
- Undo/redo in TRichViewEdit[54].

1.5.1.2.45 TRVTableItemInfo.SetCellOptions

Assigns **Value** to Cells[537][**Row**, **Col**].Options[571], as an editing operation.

```
procedure SetCellOpacity(Value: TRVTableCellOptions[571]; Row,Col: Integer);
```

(introduced in version 16)

Unlike a direct assignment to Cells[537][**Row**, **Col**].Options[571] property, this method can be undone/redone, if it was called for the table inserted in TRichViewEdit (after insertion[320]).

Use this method in conjunction with methods for updating editor window, see example[122].

See also:

- Undo/redo in tables[121];
- Undo/redo in TRichViewEdit[54].

1.5.1.2.46 TRVTableItemInfo.SetCellRotation

Assigns **Value** to Cells[537][**Row**, **Col**].Rotation[573], as an editing operation.

```
procedure SetCellRotation(Value: TRVDocRotation[626]; Row,Col: Integer);
```

(introduced in version 14)

Unlike a direct assignment to Cells[537][**Row**, **Col**].Rotation[573] property, this method can be undone/redone, if it was called for the table inserted in TRichViewEdit (after insertion[320]).

Use this method in conjunction with methods for updating editor window, see example[122].

See also:

- Undo/redo in tables[121];
- Undo/redo in TRichViewEdit[54].

1.5.1.2.47 TRVTableItemInfo.SetCellTag

Assigns **Value** to Cells[537][**Row**, **Col**].Tag[574], as an editing operation.

```
procedure SetCellVAlign(Value: Tag[574]; Row,Col: Integer);
```

(introduced in version 14)

Unlike a direct assignment to Cells[537][**Row**, **Col**].Tag[574] property, this method can be undone/redone, if it was called for the table inserted in TRichViewEdit (after insertion[320]).

Use this method in conjunction with methods for updating editor window, see example[122].

See also:

- Undo/redo in tables[121];
- Undo/redo in TRichViewEdit[54]
- Tags[34]

1.5.1.2.48 TRVTableItemInfo.SetCellVAlign

Assigns **Value** to Cells[537][**Row**, **Col**].VAlign[575], as an editing operation.

```
procedure SetCellVAlign(Value: TRVCellVAlign[625]; Row,Col: Integer);
```

(introduced in version 1.6)

Unlike a direct assignment to Cells[537][**Row**, **Col**].VAlign[575] property, this method can be undone/redone, if it was called for the table inserted in TRichViewEdit (after insertion[320]).

Use this method in conjunction with methods for updating editor window, see example[122].

See also:

- Undo/redo in tables[121];
- Undo/redo in TRichViewEdit[54].

1.5.1.2.49 TRVTableItemInfo.SetCellVisibleBorders

Assigns values to Cells[537][**Row**, **Col**].VisibleBorders[575], as an editing operation.

```
procedure SetCellVisibleBorders(Left, Top, Right, Bottom: Boolean;
  Row,Col: Integer);
```

(introduced in version 1.6)

Unlike direct assignment to Cells[537][**Row**, **Col**].VisibleBorders[575] subproperties, this method can be undone/redone, if it was called for the table inserted in TRichViewEdit (after insertion[320]).

Use this method in conjunction with methods for updating editor window, see example[122].

See also:

- Undo/redo in tables[121];
- Undo/redo in TRichViewEdit[54].

1.5.1.2.50 TRVTableItemInfo.SetRow*

Assigns **Value** to Rows[543][**Row**].VAlign[578] (or PageBreakBefore[578], or KeepTogether[577]), as an editing operation.

```
procedure SetRowVAlign(Value: TRVCellVAlign[625]; Row: Integer);
procedure SetRowPageBreakBefore(Value: Boolean; Row: Integer);
procedure SetRowKeepTogether(Value: Boolean; Row: Integer);
```

(introduced in version 1.6 and 14)

Unlike direct assignments to a Rows[543][**Row**] property, these methods can be undone/redone, if they were called for a table inserted in TRichViewEdit (after insertion[320]).

Use these methods in conjunction with the methods for updating an editor window, see the example 122.

See also:

- Undo/redo in tables 121;
- Undo/redo in TRichViewEdit 54.

1.5.1.2.51 TRVTableItemInfo.SetTableVisibleBorders

Assigns values to VisibleBorders 544, as an editing operation.

```
procedure SetTableVisibleBorders(Left, Top, Right, Bottom: Boolean);
```

(introduced in version 10)

Unlike direct assignment to VisibleBorders 544 subproperties, this method can be undone/redone, if it was called for the table inserted in TRichViewEdit (after insertion 320).

Use this method in conjunction with methods for updating editor window, see example 122.

See also:

- Undo/redo in tables 121;
- Undo/redo in TRichViewEdit 54.

1.5.1.2.52 TRVTableItemInfo.SplitSelectedCellsHorizontally

Splits each selected cell into **RowCount** cells (inserting horizontal splitters).

```
procedure SplitSelectedCellsHorizontally(RowCount: Integer);
```

This method splits cells calling UnmergeCells 564, InsertRows 555 and MergeCells 556. This method can change number of rows.

It's recommended to call this method only when selection has a rectangular shape, see CanMergeSelectedCells 548.

If no cells are selected, it splits the cell that is being edited.

This method must be called when the document is formatted.

See also:

- Undo note.

See also methods:

- SplitSelectedCellsVertically 563;
- UnmergeSelectedCells 564;
- GetNormalizedSelectionBounds 553.

1.5.1.2.53 TRVTableItemInfo.SplitSelectedCellsVertically

Splits each selected cell into **ColCount** cells (inserting vertical splitters).

```
procedure SplitSelectedCellsVertically(ColCount: Integer);
```

This method splits cells calling UnmergeCells 564, InsertCols 554 and MergeCells 556. This method can change number of columns.

It's recommended to call this method only when selection has a rectangular shape, see CanMergeSelectedCells 548.

If no cells are selected, it splits the cell that is being edited.

This method must be called when the document is formatted.

See also:

- Undo note.

See also methods:

- SplitSelectedCellsVertically 563;
- UnmergeSelectedCells 564;
- GetNormalizedSelectionBounds 553.

1.5.1.2.54 TRVTableItemInfo.UnmergeCells

Unmerges cells.

```
procedure UnmergeCells(TopRow, LeftCol, ColSpan, RowSpan: Integer; UnmergeRows, UnmergeCols
```

Cells can be merged using the methods MergeCells 556 or MergeSelectedCells 557.

This method unmerges all cells in range from Cells 537[**TopRow**, **LeftCol**] to Cells 537[**TopRow**+**RowSpan**, **LeftCol**+**ColSpan**].

If **UnmergeRows**=*True*, it unmerges each cell into RowSpan 574 rows.

If **UnmergeCols**=*True*, it unmerges each cells into ColSpan 570 columns.

UnmergeRows and *UnmergeCells* can both be *True*.

See also:

- Undo note.

See also methods:

- UnmergeSelectedCells 564;
- MergeCells 556.

1.5.1.2.55 TRVTableItemInfo.UnmergeSelectedCells

Unmerges the selected cells.

```
procedure UnmergeSelectedCells(UnmergeRows, UnmergeCols: Boolean);
```

Cells can be merged using the methods MergeCells 556 or MergeSelectedCells 557.

If **UnmergeRows**=*True*, it unmerges each selected cell into RowSpan 574 rows.

If **UnmergeCols**=*True*, it unmerges each selected cells into ColSpan 570 columns.

UnmergeRows and **UnmergeCells** can both be *True*.

This method must be called when the document is formatted.

See also:

- Undo note.

See also methods:

- UnmergeCells 564;
- MergeSelectedCells 557;
- GetNormalizedSelectionBounds 553.

1.5.1.3 Events

1.5.1.3.1 TRVTableItemInfo.OnCellEditing

Occurs before starting editing of this cell and allows to prevent it.

```
type
  TRVCellEditingEvent = procedure (Sender: TRVTableItemInfo;
    Row, Col : Integer; Automatic: Boolean;
    var AllowEdit: Boolean) of object;
```

(introduced in version 1.7)

Input parameters

This event occurs when editor attempts to change content of **Sender**.Cells 537[**Row**, **Col**].

If **Automatic**=*True*, this is not actually a cell editing, but a multicell operation (clearing, applying style, etc.)

Output parameters

Set **AllowEdit** to *False* to disallow this operation.

Tables are not components, so their events cannot be assigned at design time in the Object Inspector. They must be assigned in code. Events are not saved in RVF [59] files, so you need to reassign them when tables are loaded. The best place to assign this event is TRichView.OnItemAction event.

Example

```
// This procedure is assigned to OnCellEditing of all tables
procedure TMyForm.DoCellEditing(Sender: TRVTableItemInfo; Row, Col: Integer;
  Automatic: Boolean; var AllowEdit: Boolean);
begin
  // Preventing editing all cells in the top rows of all tables
  if Row=0 then
    AllowEdit := False;
end;

// MyRichViewEdit.OnItemAction. Assigns DoCellEditing to
// OnCellEditing events of all tables inserted in MyRichViewEdit.
// (this assignment occurs when table is inserted using
// AddItem [166] or InsertItem [320] methods, or is loaded from RVF or RTF file)
procedure TMyForm.MyRichViewEditItemAction(Sender: TCustomRichView;
  ItemAction: TRVItemAction; Item: TCustomRVItemInfo [526]; var Text: String;
  RVData: TCustomRVData);
begin
  if (Item.StyleNo = rvsTable) and (ItemAction = rviaInserting) then
    TRVTableItemInfo [528](Item).OnCellEditing := DoCellEditing;
end;
```

See also:

- TParaInfo.Options [453] (*rvpaoReadOnly, rvpaoStyleProtect, rvpaoDoNotWantReturns* – protection options for paragraphs);
- TFontInfo.Protection [439] (protection options for text);
- TRVExtraItemProperty [627] (*rvepDeleteProtect, rvepResizable* – protection option for nontext items).

1.5.1.3.2 TRVTableItemInfo.OnDraw Border

Allows custom drawing of table borders

```
type
  TRVTableDrawBorderEvent = procedure (Sender: TRVTableItemInfo;
    Canvas: TCanvas; Left,Top,Right,Bottom: Integer;
    Width: TRVStyleLength [644];
    LightColor, Color, BackgroundColor: TColor;
    Style: TRVTableBorderStyle [645]; Printing: Boolean;
    VisibleBorders: TRVBooleanRect [611]; Row, Col: Integer;
    var DoDefault: Boolean) of object;

property OnDrawBorder: TRVTableDrawBorderEvent;
```

(introduced in version 1.7, modified in 1.8)

Input parameters

Sender – the table to draw border for (border can be drawn for the table itself or for the table cells).

Canvas – canvas where to draw.

Left, **Top**, **Right**, **Bottom** – rectangle for the border.

Width – border width, in **Sender**.Style[156].Units[404].

LightColor, **Color**, **BackgroundColor** – colors for border and for background (be careful - cell border is drawn after drawing cell content)

Style – border style.

Printing – *True* if this drawing is for printing; *False* if on screen.

VisibleBorders – VisibleBorders[544] property for the table, or VisibleBorders[575] property for the cell.

Row, **Col** – row and column of the cell (or -1 if this is a table border).

Output parameters

DoDefault – set to *False* to cancel the default border drawing.

For example,

```
{ Drawing a hairline border (both on screen and printer),
  assuming that CellBorderWidth=1, CellVSpacing=0, CellHSpacing=0.
  This example ignores VisibleBorders.
}
procedure TMyForm.DoDrawBorder(Sender: TRVTableItemInfo; Canvas: TCanvas;
  Left, Top, Right, Bottom: Integer; Width: TRVStyleLength;
  LightColor, Color, BackgroundColor: TColor; Style: TRVTableBorderStyle;
  Printing: Boolean; VisibleBorders: TRVBooleanRect; var DoDefault: Boolean);
begin
  if Width=0 then
    exit;
  inc(Right);
  inc(Bottom);
  Canvas.Pen.Color := Color;
  Canvas.Pen.Style := psInsideFrame;
  Canvas.Pen.Width := 1;
  Canvas.Brush.Style := bsClear;
  Canvas.Rectangle(Left,Top,Right,Bottom);
  DoDefault := False;
end;
```

Tables are not components, so their events cannot be assigned at design time in the Object Inspector. They must be assigned in code. Events are not saved in RVF[59] files, so you need to reassign them when tables are loaded. The best place to assign this event is TRichView.OnItemAction event.

```
// MyRichViewEdit.OnItemAction. Assigns DoDrawBorder to
// OnDrawBorder events of all tables inserted in MyRichViewEdit.
// (this assignment occurs when table is inserted using
// AddItem[166] or InsertItem[320] methods, or is loaded from RVF or RTF file)
procedure TMyForm.MyRichViewEditItemAction(Sender: TCustomRichView;
  ItemAction: TRVItemAction; Item: TCustomRVItemInfo[526]; var Text: String;
  RVData: TCustomRVData);
begin
  if (Item.StyleNo = rvsTable) and (ItemAction = rviaInserting) then
    TRVTableItemInfo[528](Item).OnDrawBorder := DoDrawBorder;
end;
```

1.5.1.4 Classes of Properties

1.5.1.4.1 TRVTableCellData

TRVTableCellData represents one cell in a table (see TRVTableItemInfo[528]). Cells can be accessed using Cells[537] property of table.

Unit RVTable.

Syntax

```
TRVTableCellData = class(TRVItemFormattedData[608])
```

Hierarchy

TObject
TPersistent
TCustomRVData[608]
TCustomRVFormattedData
TRVItemFormattedData

Methods and Properties

TRVTableCellData has many properties and methods corresponding to properties and methods[576] of TRichView[128].

TRVTableCellData and class of object TRichView.RVData[151] are derived from the same base class.

In addition, this class has properties affecting the cell view and layout:

- Color[570] – background color of cell;
- BackgroundImage[567], BackgroundStyle[568] – background image and its position;
- BackgroundImageFileName[568] – can be used when exporting to HTML;
- BorderColor[569], BorderLightColor[570] – colors of border;
- BestWidth[569], BestHeight[568] – preferred size of cell,
- VAlign[575] – vertical alignment of contents;
- Rotation[573] – rotation of contents;
- VisibleBorders[575] – defines visible sides of cell border.
- Tag[574] – a string value (not used by the component itself).

These properties can be saved with cell in RVF[59] file or stream.

Also, this class has read-only properties:

- ColSpan[570], RowSpan[574].

Important methods:

- GetRVData[575].

See also

Table Cells Overview[113]

1.5.1.4.1.1 Properties

Background image for the table cell

```
property BackgroundImage: TGraphic;
```

(introduced in v1.8)

Assignment to this property copies the image, so you still need to free the source image yourself.

Example 1 (assigning image):

```
bmp := TBitmap.Create;
bmp.LoadFromFile('c:\image.bmp');
table.Cells[r,c].BackgroundImage := bmp;
```

```
bmp.Free;
```

Example 2 (clearing the image):

```
table.Cells[r,c].BackgroundImage := nil;
```

See also properties:

- BackgroundImageFileName 568;
- BackgroundStyle 568;
- Color 570.

See also properties of table:

- BackgroundImage 530;
- BackgroundStyle 533.

See also properties of TRichView:

- BackgroundBitmap 136;
- BackgroundStyle 136.

This is a string for storing file name associated with BackgroundImage 567

```
property BackgroundImageFileName: String;
```

(introduced in v1.9)

If *rvsoUseItemImageFileNames* is included in the Options parameter of SaveHTML 216/SaveHTMLEx 217, images with defined (non-empty) file names will not be saved, but their file names will be written in HTML (unlike pictures and tables, cells are not items, so this property cannot be accessed using item string properties 630).

See also properties of table:

- BackgroundImageFileName 531.

Defines the position of BackgroundImage 567.

```
property BackgroundStyle: TRVItemBackgroundStyle 636;
```

(introduced in v1.8)

See also:

- BackgroundImage 567;
- Color 570.

See also properties of table:

- BackgroundStyle 533;
- BackgroundImage 530.

See also properties of TRichView:

- BackgroundStyle 136;
- BackgroundBitmap 136.

Minimal height for the cell.

```
property BestHeight: TRVStyleLength 644;
```

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units 404 of the linked TRVStyle 385 component.

This property corresponds to <td height> in HTML.

A direct assignment to this property cannot be undone/redone by user, use table.SetCellBestHeight 559.

The cell can be higher if it's required to show all its content (if *rvtoIgnoreContentHeight* is not included in table.Options 540).

Default value:

0

See also properties:

- BestWidth 569.

See also properties of table:

- Options 540 (*rvtoIgnoreContentHeight* option).

See also:

- Table Resizing 124.

Preferred width for the cell

```
property BestWidth: TRVHTMLLength 635;
```

This is a preferred width for the cell, measured either in Units 404 of the linked TRVStyle 385 component (pixels or twips) or in percent.

It corresponds to <td width> in HTML.

A direct assignment to this property cannot be undone/redone by user, use table.SetCellBestWidth 560.

The cell can be wider than this value, if it's required to display its contents without overlapping. Cell always has enough width for displaying the widest its item + CellHPadding 536*2 + paragraph indents (FirstIndent 451, LeftIndent 451, RightIndent 454) (if *rvtoIgnoreContentWidth* is not included in table.Options 540).

The cell can be narrower than specified in this property, if the table is not wide enough.

Default value:

0

See also properties:

- BestHeight 568.

See also properties of table:

- BestWidth 531;
- Options 540 (*rvtoIgnoreContentWidth* option).

See also:

- Table Resizing 124.

Color of border for this cell.

```
property BorderColor: TColor
```

(introduced in version 1.6)

If this value of this property is *clNone*, the cell uses table.CellBorderColor 534.

This value defines color of flat border around the cell or "dark" color of 3d border.

A direct assignment to this property cannot be undone/redone by user, use table.SetCellBorderColor 560.

Default value:

clNone

See also properties:

- BorderLightColor 570;
- Color 570.

See also properties of table:

- CellBorderColor 534;
- BorderColor 532;
- CellBorderStyle 535.

"Light" color of 3d border for this cell

```
property BorderLightColor: TColor;
```

(introduced in version 1.6)

If this value of this property is *clNone*, the cell uses table.CellBorderLightColor[534].

This value defines light color of 3d border.

A direct assignment to this property cannot be undone/redone by user, use table.SetCellBorderLightColor[560].

Default value:

clNone

See also properties:

- BorderColor[569];
- Color[570].

See also properties of table:

- CellBorderColor[534];
- BorderColor[532];
- CellBorderStyle[535].

Background color of the cell

```
property Color: TColor;
```

Set to *clNone* for default background.

This property corresponds to <td bgcolor> in HTML.

A direct assignment to this property cannot be undone/redone by user, use table.SetCellColor[560].

If this property is equal to clNone, a color of a row/column[546] can be used.

Default value:

clNone

See also properties:

- BorderColor[569];
- BorderLightColor[570].

See also properties of table:

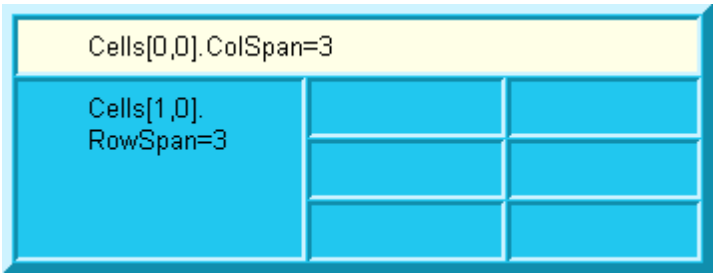
- Color[538];
- CellBorderColor[534];
- CellBorderLightColor[534].

Read-only positive integer value specifying the number of columns spanned by this cell.

```
property ColSpan: Integer;
```

Initially this property is equal to 1 for all cells. This value can be changed due to cell-merging

Example:



ColSpan Example

This property corresponds to <td colspan> in HTML.

See also:

- RowSpan[574].

Contains the text string that can appear when the user moves the mouse pointer over the cell.

```
property Hint: String;
```

(introduced in version 10)

Default value:

" (empty string)

See also:

- TRVExtraItemStrProperty[630] (*rvespHint* defines items hints, including table hints);
- TCustomRichView.OnItemHint[246] (unlike item hints, cell hints cannot be altered in this event).

Opacity of cell background

```
property Opacity: TRVOpacity[637];
```

(introduced in version 16)

This value is used if the cell color<>*clNone*. The opacity is applied only to the background color filling, not to the background image.

A direct assignment to this property cannot be undone/redone by user, use table.SetCellOpacity[561].

The cell color is specified in Color[570] property. If Color[570]=*clNone*, the default row/column color[546] is used.

Default value:

100000 (i.e. 100%)

See also properties of table:

- Opacity[540]

Options for table cell

```
type
  TRVTableCellOption = (rvtcoReverseLineOrder, rvtcoUseVerticalFonts);
  TRVTableCellOptions = set of TRVTableCellOption;
property Options: TRVTableCellOptions;
```

(introduced in version 16)

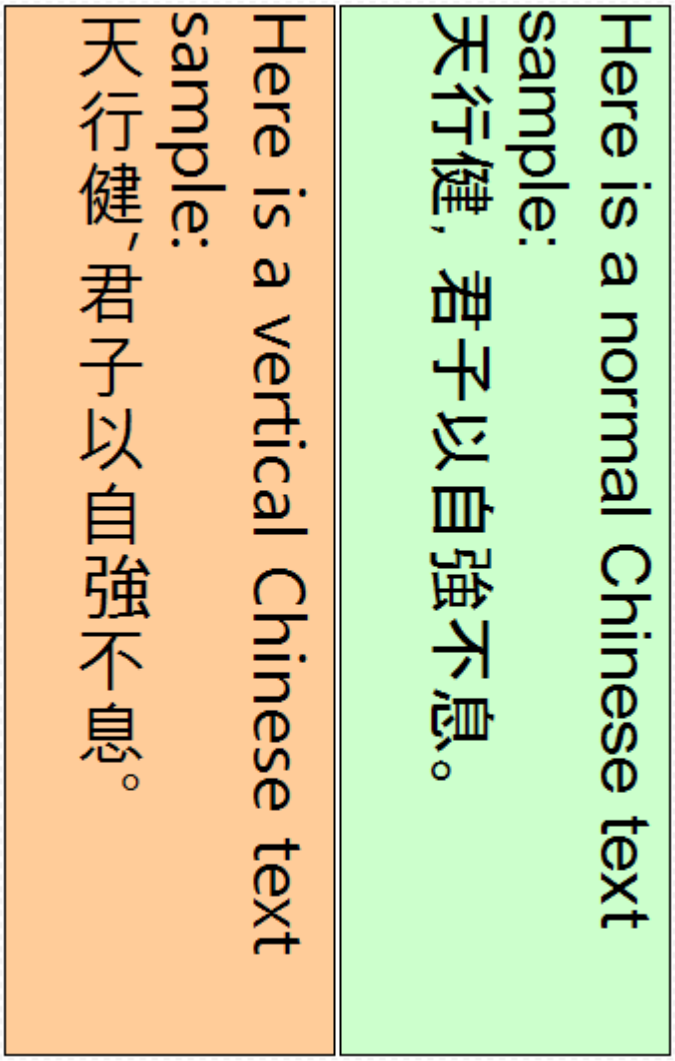
Option	Meaning
rvtcoReverseLineOrder	If set, lines in this cell are have reverse order. For example: • if Rotation[573]=*rvrotNone*, lines are ordered from bottom to top; • if Rotation[573]=*rvrot90*, lines are ordered from left to right
rvtcoUseVerticalFonts	If set, the component adds '@' to the beginning of font names to when applying fonts to canvas, if Rotation[573] is *rvrot90* or *rvrot270.*

In vertical fonts, East Asian characters are rotated by 90° counterclockwise, so, if the cell is rotated by 90° clockwise, these characters are not rotated.

Primary meaning of these options is supporting vertical East Asian scripts.

For example:

- For vertical Chinese, Japanese or Korean text, use Rotation[573]=*rvrot90* and Options=[*rvtcoUseVerticalFonts*]. In these scripts, characters in vertical and horizontal writing must have the same orientation, columns are ordered from right to left. On the image below, the left cell has this option included.



- For vertical traditional Mongolian text, use Rotation[573]=*rvrot90*, VAlign[575]=*rvcBottom*, Options=[*rvtcoReverseLineOrder*]. In this script, columns are ordered from left to right.

Default value:

[]

Defines the rotation of the cell content.

```
property Rotation:TRVDocRotation[626];
```

(introduced in version 14)

Rotates the cell content by the specified degree.

A direct assignment to this property cannot be undone/redone by user, use table.SetCellRotation[561].

A content rotated by 90° and by 270° does not affect to width of table columns and height of table rows (if it goes beyond the cell, it is clipped).

A rotation by 90° can be used to implement vertical text, including East Asian scripts. Use it in conjunction with Options[571].

Limitations:

- this property is ignored when the application is run in Windows 95, 98, or ME;

- TPngObject/TPngImage cannot show transparency correctly if rotated;
- transparency limitation when printing (you cannot see a bitmapped background through a rotated picture further than the background of its table) (this limitation does not apply to ScaleRichView);
- no corrections are made to drawing procedures to compensate differences in drawing graphics in the standard and the advanced modes (when drawing rotated content, Canvas is switched to the advanced graphic mode);
- controls[94] inserted in rotated cells are not shown (this limitation does not apply to ScaleRichView).

Saving and loading:

- this property can be saved and loaded from RVF[59];
- rotations by 90° and by 270° are saved to RTF as the corresponding text flow in the cell, and are loaded accordingly;
- this property is not saved and not read from HTML.

See also notes about rotation in the topics about:

- Cell.VAlign[575] property;
- Table.CellPadding[536]

Default value:

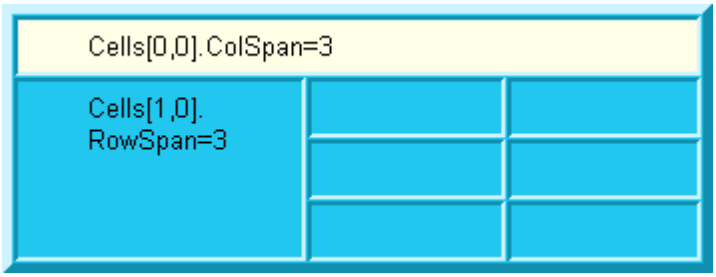
rvrotNone

Read-only positive integer value specifying the number of rows spanned by this cell

```
property RowSpan: Integer;
```

Initially this property is equal to 1 for all cells. This value can be changed due to cell-merging.

Example:



RowSpan Example

This property corresponds to <td rowspan> in HTML.

See also:

- ColSpan[570].

Contains additional string value associated with this table cell.

```
property Tag: TRVTag[645];
```

(introduced in version 14)

This value is not used by the component itself. *Tags* are provided for the convenience of storing additional string value for special needs in your application.

Default value:

'' (empty string)

See also:

- Tags[34]

Defines vertical alignment of the cell content.

```
property VAlign: TRVCellVAlign[625];
```

(introduced in version 1.6)

If this property is set to *rvcVDefault*, the cell uses vertical alignment of its row, see TRVTableRow.VAlign[578].

A direct assignment to this property cannot be undone/redone by user, use table.SetCellVAlign[562].

If the cell is rotated[573], VAlign works relative to the cell content, not relative to the cell. For example, if the cell is rotated by 90°, *rvcTop* aligns the cell content to the right side, *rvcMiddle* centers vertically, *rvcBottom* aligns to the left side.

Default value:

rvcVDefault

Allows to hide some sides of cell border.

```
property VisibleBorders: TRVBooleanRect[611];
```

A direct assignment to this property cannot be undone/redone by user, use table.SetCellVisibleBorders[562].

Default value:

True, *True*, *True*, *True* (all sides are visible)

See also properties of table:

- VisibleBorders[544];
- CellBorderWidth[535];
- CellBorderColor[534];
- CellBorderLightColor[534].

1.5.1.4.1.2 Methods

Returns object containing cell data (document items)

```
function GetRVData: TCustomRVData[608];
```

The cell is "an object containing data" itself. But when cell is in editing state[117], all its data are moved to inplace editor, and the cell is empty.

This method returns:

- the cell itself, if it is not in editing state;
- RVData[151] property of inplace editor, otherwise.

So when working with content of cell directly[576], you can use this method and do not worry about editing state of a cell.

Example (retrieving text from table):

```
function GetTableText(table: TRVTableItemInfo): String;
var
  i, r, c: Integer;
  RVData: TCustomRVData[608];
begin
  Result := '';
  for r := 0 to table.RowCount[543]-1 do
  begin
    for c := 0 to table.ColCount[538]-1 do
      if table.Cells[537][r,c]<>nil then
      begin
        RVData := table.Cells[r,c].GetRVData;
        for i := 0 to RVData.ItemCount[144]-1 do
        begin
          if (i>0) and RVData.IsFromNewLine[207](i) then
```

```
            Result := Result + #13#10;
          if RVData.GetItemStyle[194](i)=rvsTab then
            Result := Result + #9
           else if RVData.GetItemStyle(i)>=0 then
            Result := Result + RVData.GetItemText[195](i);
        end;
        Result := Result + table.TextColSeparator[543];
      end;
    Result := Result + table.TextRowSeparator[544];
  end;
end;
```

See also methods:

- GetSourceRVData[611].

1.5.1.4.1.3 Additional properties of TRVTableCellData

TRVTableCellData[567] has many properties and methods, similar to properties and methods of TRichView[128].

Some of them are listed below.

But when the cell is in editing state, RichViewEdit[282] creates special inplace-editor and temporary moves all cell content to this editor.

To provide correct work with cell content both in normal and in editing state, call methods of cell.GetRVData[575] instead of methods of the cell itself.

When working in editor, you seldom need to access cells' data directly, the most of operations can be performed using the methods that work with item at the caret position, read more[112].

See also: TRVTableItemInfo.Changed[548]

Cell methods for appending items

(links are to the corresponding methods of TCustomRichView)

- AddBreak, AddBreakEx, AddBreakExTag, AddBreakTag[160];
- AddBulletEx, AddBulletExTag[160];
- AddControlEx, AddControlExTag[162];
- AddHotspotEx, AddHotspotExTag[164];
- AddFmt[163], AddItem[166];
- AddNL, AddNLTag[167], AddTextNL[171];
- AddPictureEx, AddPictureExTag[169];
- AddNLATag, AddNLWTag[167].

(note: it's not recommended to use the methods of the group above for cells of table inserted in TRichViewEdit[282], if editor's UndoLimit[295]<>0)

Obtaining information from cell

- GetItemStyle[194];
- GetBreakInfo[183], GetBulletInfo[184], GetControlInfo[187], GetHotspotInfo[189], GetPictureInfo[201], GetTextInfo[204];
- GetItemTag[195], IsParaStart[208], GetItemPara[194], IsFromNewLine[207], GetOffsBeforeItem[201], GetOffsAfterItem[200].

Modifying items

SetBreakInfo[226], SetBulletInfo[227], SetControlInfo[228], SetHotspotInfo[229], SetPictureInfo[234], SetItemTag[231].

(note: it's not recommended to use the methods of the group above for cells of table inserted in TRichViewEdit[282], if editor's UndoLimit[295]<>0)

1.5.1.4.2 TRVTableRow

This class represents one row in table (TRVTableItemInfo) 528

Unit RVTable;

Syntax

```
TRVTableRow = class(TRVDataList)
```

Hierarchy

TObject
TList
TRVList
TRVDataList

Using

Table has Rows 543 property of class TRVTableRows 578. TRVTableRows is a list of objects of class TRVTableRow. TRVTableRow object is rarely, if ever, created directly in an application. It used only as an item of Rows property of tables.

Please do not use methods of this class for modifying (adding, deleting, replacing) cells. Use the proper methods of TRVTableItemInfo 528 instead.

Properties

The key properties of this class are Items 577 and Count. Count is the number of columns in the table (all rows in the table have the same number of cells).

VAlign 578 defines the default vertical alignment for the table cells in this row.

PageBreakBefore 578 allows adding a page break before this row.

KeepWithNext 577 allows keeping the content of this row on a single page, if possible.

See Also

Table Cells Overview 113

1.5.1.4.2.1 Properties

Provides indexed access to the cells of table

```
property Items[Index: Integer]:TRVTableCellData 567; default;
```

Each item represents one cell of the table.

Usually cells are accessed as Cells 537 property of table instead.

Allows to keep the content of this row on the same page, if possible.

```
property KeepTogether: Boolean;
```

(introduced in version 14)

This property is ignored if *rvtoRowsSplit* is not included in table.PrintOptions 542.

The effect is similar to including *rvpaoKeepWithNext* in Options 453 of all paragraphs of all cells of this row.

A direct assignment to this property cannot be undone/redone by the user, use table.SetRowKeepTogether 562.

Default value:

False

Allows to set explicit page break before this row.

```
property PageBreakBefore: Boolean;
```

(introduced in version 10)

If at least one row has PageBreaksBefore=*True*, an absence of *rvtoRowsSplit* in table.PrintOptions[542] is ignored.

A direct assignment to this property cannot be undone/redone by the user, use table.SetRowPageBreakBefore[562].

Default value:

False

Defines default vertical alignment for contents of cells in the row

```
property VAlign:TRVCellVAlign[625];
```

(introduced in version 1.6)

rvcVDefault works like *rvcMiddle*.

A direct assignment to this property cannot be undone/redone by the user, use table.SetRowVAlign[562].

Default value:

rvcTop

See also properties of cell:

- VAlign[575]

1.5.1.4.3 TRVTableRows

TRVTableRows represents contents of table (see TRVTableItemInfo[528])

Unit RVTable.

Syntax

```
TRVTableRows = class(TRVList)
```

Hierarchy

TObject
TList
TRVList

Using

This is the type of Rows[543] properties of table. TRVTableRows is a list of TRVTableRow[577]s which are in their order are lists of TRVTableCellData[567] (i.e. table cells)

TRVTableRows object is rarely, if ever, created directly in an application. It used only as Rows property of tables.

Please do not use methods of this class for modifying (adding, deleting, replacing) rows. Use the proper methods of TRVTableItemInfo[528] instead.

The key properties of this class are Items[579] and Count. Count is the number of rows in the table.

This class has method for finding main cell for given merged cell: GetMainCell[579].

See Also

Table Cells Overview[113]

1.5.1.4.3.1 Properties

Provides indexed access to the rows of table

```
property Items[Index: Integer]: TRVTableRow[577]; default;
```

Each item represents one row of the table.

1.5.1.4.3.2 Methods

Returns the main (top left) cell for given merged cell.

```
function GetMainCell(ARow,ACol: Integer; var MRow, MCol: Integer): TRVTableCellData[567];
```

Some cells[537] in table[528] can be equal to *nil* due to cell-merging. This method allows to find the top left cell overlapping the given merged cell.

Input parameters:

ARow and **ACol** specify some cell in the table. Table.Cells[537][**ARow**, **ACol**] can be *nil*.

Output parameters:

MRow and **MCol** receive position of the main cell. This cell is never equal to *nil*.

if Table.Cells[**ARow**, **ACol**]<>*nil* then **MRow** will be equal to **ARow** and **MCol** will be equal to **ACol**.

Return value:

Table.Cells[537][**MRow**, **MCol**]

1.5.2 Label - TRVLabelItemInfo

TRVLabelItemInfo is a class representing "label"[101] in TRichView documents. This is a non-text item looking like text (but it cannot be wrapped). This is not a component. Objects of this class are created at runtime and appended to TRichView (see AddItem[166]) or inserted into TRichViewEdit (see InsertItem[320]). Style[194] of this item type: *rvsLabel* (-200)

Unit RVLabelItem.

Syntax

```
TRVLabelItemInfo = class(TRVRectItemInfo)
```

(introduced in version 10)

Hierarchy

TObject
TPersistent
TCustomRVItemInfo[526]
TRVNonTextItemInfo
TRVRectItemInfo

Properties

This item displays Text[581] using font defined in TextStyleNo[581]. ProtectTextStyleNo[581] can be used to protect TextStyleNo[581] from changing.

You can set minimal possible width for this item (MinWidth[580]) and horizontal text Alignment[580].

You can define Cursor[580] for this item.

Use SetCurrentItemExtraIntPropertyEx[345] and SetCurrentItemExtraStrPropertyEx[346] to change properties of label items as editing operations.

Related Properties of TRVStyle [385]

- FieldHighlightColor [390] – color for highlighting label items (overrides background color [433] specified in the text style)
- FieldHighlightType [390] specifies when to highlight label items.

Inherited Classes

The following item types are inherited from this class:

- TRVSeqItemInfo [582] (numbered sequences);
- TRVEndNoteItemInfo [587] (endnote);
- TRVFootnoteItemInfo [589] (footnote);
- TRVNoteReferenceItemInfo [602] (reference to the parent footnote or endnote).

See Also...

Demos:

- Demos\Delphi\Assorted\Fields\MailMerge-LabelItems\
- Demos\CBuilder\Assorted\Fields\MailMerge-LabelItems\

1.5.2.1 Properties

1.5.2.1.1 TRVLabelItemInfo.Alignment

Defines a horizontal alignment of text relative to the item.

```
property Alignment: TAlignment;
```

This property makes sense if Text [581] width is less than MinWidth [580].

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx [345] to change value of this property as an editing operation.

Default value:

taLeftJustify

1.5.2.1.2 TRVLabelItemInfo.Cursor

Cursor for the item.

```
Cursor: TCursor;
```

Value of this property is not stored in RVF [59].

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx [345] to change value of this property as an editing operation.

Default value:

crDefault (using default cursor)

1.5.2.1.3 TRVLabelItemInfo.MinWidth

Defines a minimal width of this item.

```
property MinWidth: Integer;
```

This value is measured either in screen pixels (when printing, it is recalculated according to the printer resolution) or in twips, depending on Units [404] of the linked TRVStyle [385] component.

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx [345] to change value of this property as an editing operation.

Default value:

0

See also properties:

- Alignment 580;
- Text 581.

1.5.2.1.4 TRVLabelItemInfo.ProtectTextStyleNo

If set to *True*, TCustomRichViewEdit.ApplyTextStyle 300 and ApplyStyleConversion 299 will not be able to change TextStyleNo 581 of this item.

```
ProtectTextStyleNo: Boolean;
```

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx 345 to change value of this property as an editing operation.

Default value:

False

1.5.2.1.5 TRVLabelItemInfo.RemoveInternalLeading

Allows removing spacing above the text.

```
RemoveInternalLeading: Boolean;
```

If *True*, the label item is displayed without leading (space) above the text (i.e. without font internal leading). Accent marks and other diacritical characters may occur in this area, so do not use this property if Text 581 contains accented characters.

This property is useful is this label item is used as a drop cap (the first letter in a chapter, with vertical alignment 648 = *rvvaLeft*).

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx 345 to change value of this property as an editing operation.

Default value:

False

1.5.2.1.6 TRVLabelItemInfo.Text

Visible text

```
Text: String;
```

Unlike text items 88, this is not the text returned by TCustomRichView.GetItemText 195.

Unicode note: For Delphi/C++Builder 2009+, this is a Unicode 64 text. For older versions, this is ANSI text.

Use TRichViewEdit.SetCurrentItemExtraStrPropertyEx 346 to change value of this property as an editing operation.

See also properties:

- Alignment 580;
- TextStyleNo 581.

1.5.2.1.7 TRVLabelItemInfo.TextStyleNo

Defines text attributes.

```
property TextStyleNo: Integer;
```

This is the index in the TextStyles 404 collection of the TRVStyle 385 component linked to the TCustomRichView 128 component containing this item.

Unicode 446 property of text style is ignored.

If this item is inserted in editor 282, value of this property may be changed when ApplyTextStyle 300 or

ApplyStyleConversion[299] method is called. You can prevent this by assigning *True* to ProtectTextStyleNo[581] property.

If style templates[158] are used, the component may automatically change value of this property to provide a consistency of text and paragraph style templates (even if ProtectTextStyleNo[581]=*True*).

See also properties:

- Text[581].

1.5.2.2 Methods

1.5.2.2.1 TRVLabelItemInfo.Create

A constructor, creates a new TRVLabelItemInfo object

```
constructor Create(RVData: TPersistent); override;
```

Parameter:

RVData – RVData[151] property of TRichView[128] or TRichViewEdit[282], where you wish to insert this item.

This constructor creates a new label with empty Text[581] and TextStyleNo[581]=0.

There is more convenient constructor, CreateEx[582].

See also methods of TCustomRichView:

- AddItem[166].

See also methods of TCustomRichViewEdit:

- InsertItem[320].

1.5.2.2.2 TRVLabelItemInfo.CreateEx

A constructor, creates a new TRVLabelItemInfo object with the specified Text[581] and TextStyleNo[581].

```
constructor CreateEx(RVData: TPersistent;
  TextStyleNo: Integer; const Text: String);
```

Parameters:

RVData – RVData[151] property of TRichView[128] or TRichViewEdit[282] where you wish to insert this item.

TextStyleNo – value for TextStyleNo[581] property.

Text – value for Text[581] property.

See also methods:

- Create[582].

See also methods of TCustomRichView:

- AddItem[166].

See also methods of TCustomRichViewEdit:

- InsertItem[320].

1.5.3 Numbered Sequence - TRVSeqItemInfo

TRVSeqItemInfo is a class representing numbered sequence[102] in TRichView documents. This is not a component. Objects of this class are created at runtime and appended to TRichView (see AddItem[166]) or inserted into TRichViewEdit (see InsertItem[320]). Style[194] of this item type: *rvsSequence* (-202)

Unit RVSeqItem.

Syntax

```
TRVSeqItemInfo = class(TRVLabelItemInfo[579])
```

(introduced in version 10)

Hierarchy

TObject
TPersistent
TCustomRVItemInfo[526]
TRVNonTextItemInfo
TRVRectItemInfo
TRVLabelItemInfo[579]

Properties

The main property of this class is SeqName[585]. All items having the same value of this property are numbered continuously. You can restart numbering from the specified value, see StartFrom[585] and Reset[584] properties.

Numbering type is defined in NumberType[584] property.

Font for this item is defined in TextStyleNo[581]. ProtectTextStyleNo[581] can be used to protect TextStyleNo[581] from changing.

Text[581] for this item is calculated automatically.

Use SetCurrentItemExtraIntPropertyEx[345] and SetCurrentItemExtraStrPropertyEx[346] to change properties of label items as editing operations.

Related Properties of TRVStyle[385]

- FieldHighlightColor[390] – color for highlighting numbering sequences (overrides background color[433] specified in the text style)
- FieldHighlightType[390] specifies when to highlight numbering sequences.

Numbering Sequences and List Markers

Numbered sequences are similar to list markers[99]. The table below shows differences between them.

Numbered sequences	List markers
Items can be inserted in any place of document.	List markers start paragraphs.
Counter for the item is calculated using SeqName[585] property.	Counter for the item is calculated using list index and list level.
Simple numbering.	Multilevel numbering.
Visual appearance is defined using the item properties.	Visual appearance is defined in list styles[396].
Items in the same sequence may have different fonts, numbering type, format strings.	All list markers linked to the same list level have the same font, numbering type and format string.
TCustomRichViewEdit.ApplyTextStyle[300] and ApplyStyleConversion[299] can change item font.	TCustomRichViewEdit.ApplyTextStyle[300] and ApplyStyleConversion[299] cannot change item font.

Inherited Classes

The following item types are inherited from this class:

- TRVEndNoteItemInfo 587 (endnote);
- TRVFootnoteItemInfo 589 (footnote);

See Also...

Demos:

- Demos\Delphi\Assorted\Fields\MailMerge-LabelItems\
- Demos\CBuilder\Assorted\Fields\MailMerge-LabelItems\

1.5.3.1 Properties

1.5.3.1.1 TRVSeqItemInfo.FormatString

Text format.

```
FormatString: String;
```

If value of this property is not empty, it defines format for text. Use '%s' to specify where to insert numbering.

For example, if FormatString = 'Figure %s.', NumberType 584 = *rvseqUpperRoman*, and counter value is 2, this item displays 'Figure II.'.

You can assign this property before inserting this item in TRichView.

This property is ignored when saving documents to RTF.

If you change this property for an item that was already inserted in a document, call RichView.RefreshSequences 214.

Use TRichViewEdit.SetCurrentItemExtraStrPropertyEx 346 to change value of this property as an editing operation.

Default value:

" (empty string)

1.5.3.1.2 TRVSeqItemInfo.NumberType

Numbering type.

```
property NumberType: TRVSeqType 643;
```

See TRVSeqType 643 for possible values.

This property is initialized in the constructor CreateEx 585.

This property is ignored in inherited classes: TRVFootnoteItemInfo 589, TRVEndnoteItemInfo 587, TRVSidenoteItemInfo 591, TRVTextBoxItemInfo 601.

If you change this property for an item that was already inserted in a document, call RichView.RefreshSequences 214.

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx 345 to change value of this property as an editing operation.

See also properties:

- FormatString 584.

1.5.3.1.3 TRVSeqItemInfo.Reset

Resets counter to StartFrom 585.

```
Reset: Boolean;
```

This property is initialized in constructor CreateEx 585.

If you change this property for an item that was already inserted in a document, call RichView.RefreshSequences 214.

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx 345 to change value of this property as an editing operation.

1.5.3.1.4 TRVSeqItemInfo.SeqName

Name of sequence. Items having the same value of this property are numbered continuously.

```
SeqName: String;
```

Character case is ignored for this property.

The following values are reserved and cannot be used for this property for TRVSeqItemInfo:

- '@footnote@' (used in TRVFootnoteItemInfo 589);
- '@endnote@' (used in TRVEndnoteItemInfo 587).
- '@sidenote@' (used in TRVSidenoteItemInfo 591).
- '@textbox@' (used in TRVTextBoxItemInfo 601).

This property is initialized in constructor CreateEx 585.

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx 345 to change value of this property as an editing operation.

1.5.3.1.5 TRVSeqItemInfo.StartFrom

Value of counter for this item, if Reset 584 = *True*.

```
StartFrom: Integer;
```

This property is initialized in constructor CreateEx 585.

If you change this property for an item that was already inserted in a document, call RichView.RefreshSequences 214.

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx 345 to change value of this property as an editing operation.

1.5.3.2 Methods

1.5.3.2.1 TRVSeqItemInfo.Create

A constructor, creates a new TRVSeqItemInfo object

```
constructor Create(RVData: TPersistent); override;
```

Please do not use this constructor, because it does not allow to set values of properties (including read-only properties).

Use CreateEx 585 instead.

1.5.3.2.2 TRVSeqItemInfo.CreateEx

A constructor, creates a new TRVSeqItemInfo object.

```
constructor CreateEx(RVData: TPersistent; const ASeqName: String;
  ANumberType: TRVSeqType; ATextStyleNo, AStartFrom: Integer;
  AReset: Boolean);
```

Parameters:

RVData – RVData 151 property of TRichView 128 or TRichViewEdit 282 where you wish to insert this item.

ASeqName – value for SeqName 585 property.

ANumberType – value for NumberType 584 property.

ATextStyleNo – value for TextStyleNo 581 property.

AStartFrom – value for StartFrom[585] property.

AReset – value for Reset[584] property.

See also methods of TCustomRichView:

- AddItem[166].

See also methods of TCustomRichViewEdit:

- InsertItem[320].

1.5.4 Ancestor for Notes - TCustomRVNoteItemInfo

TCustomRVNoteItemInfo is an ancestor class for footnote[104], endnote[103], sidenote[105] and text box[107] items.

Objects of this class must not be created directly, create objects of the following inherited classes:

- TRVFootnoteItemInfo[589];
- TRVEndnoteItemInfo[587];
- TRVSidenoteItemInfo[591];
- TRVTextBoxItemInfo[601].

Unit RVNote.

Syntax

```
TCustomRVNoteItemInfo = class(TRVSeqItemInfo[582])
TRVFootOrEndnoteItemInfo = class(TCustomRVNoteItemInfo)
```

(introduced in version 10 and 15)

Hierarchy

TObject
TPersistent
TCustomRVItemInfo[526]
TRVNonTextItemInfo
TRVRectItemInfo
TRVLabelItemInfo[579]
TRVSeqItemInfo[582]

Properties

This class introduces Document[586] property – footnote/endnote/sidenote/textbox text. This text is not displayed in TCustomRichView, but it is displayed on printing and in ScaleRichView.

Document[586] uses the same TRVStyle component as the main document (where a note is inserted).

You can use ReplaceDocumentEd[587] method to change Document[586] as an editing operation.

See Also...

Demos:

- Demos\Delphi\Editors\Notes
- Demos\CBuilder\Editors\Notes

1.5.4.1 Properties

1.5.4.1.1 TCustomRVNoteItemInfo.Document

Footnote/endnote text (displayed on printing)

```
property Document: TRVNoteData[608];
```

You can use ReplaceDocumentEd[587] method to change Document as an editing operation.

1.5.4.1.2 TCustomRVNoteItemInfo.NoteText

Text for assigning to NoteText[146] property of TRichViewEdit[282].

```
property NoteText: String;
```

(introduced in version 15)

This property returns Text[581].

If TRichViewEdit is used as an editor for Document[586], you should assign this property to its NoteText[146] property, to allow a proper displaying of note references[108].

1.5.4.2 Methods

1.5.4.2.1 TCustomRVNoteItemInfo.CreateEx

A constructor, creates a new TCustomRVNoteItemInfo object

```
constructor CreateEx(RVData: TPersistent;
  ATextStyleNo, AStartFrom: Integer; AReset: Boolean); virtual;
```

Do not create objects of this class. Create objects of inherited classes:

- TRVFootnoteItemInfo[589];
- TRVEndnoteItemInfo[587];
- TRVSidenoteItemInfo[591];
- TRVTextBoxItemInfo[601].

Parameters

RVData – RVData[151] property of TRichView[128] or TRichViewEdit[282] where you wish to insert this item.

ATextStyleNo – value for TextStyleNo[581] property. It's recommended to use RVGetNoteTextStyleNo[619] function for assigning this parameter.

AStartFrom – value for StartFrom[585] property.

AReset – value for Reset[584] property.

1.5.4.2.2 TCustomRVNoteItemInfo.ReplaceDocumentEd

Replaces content of Document[586] as an editing operation.

```
procedure ReplaceDocumentEd(Stream: TStream);
```

Stream must contain new document in RVF format[59].

1.5.5 Endnote - TRVEndnoteItemInfo

TRVEndnoteItemInfo is a class representing an endnote[103] in TRichView documents. This is not a component. Objects of this class are created at runtime and appended to TRichView (see AddItem[166]) or inserted into TRichViewEdit (see InsertItem[320]). Style[194] of this item type: *rvsEndnote* (-204)

Unit RVNote.

Syntax

```
TRVEndnoteItemInfo = class(TRVFootOrEndnoteItemInfo[586])
```

(introduced in version 10)

Hierarchy

TObject
TPersistent
TCustomRVItemInfo[526]
TRVNonTextItemInfo

TRVRectItemInfo
TRVLabelItemInfo 579
TRVSeqItemInfo 582
TCustomRVNoteItemInfo 586
TRVFootOrEndnoteItemInfo 586

Using

Numbering type for endnotes is defined in EndnoteNumbering 389 property of TRVStyle component (NumberType 584 property, inherited from TRVSeqItemInfo 582, is ignored).

You can enumerate all endnotes in documents using RVGetFirstEndnote and RVGetNextEndnote 618 functions.

Use RVGetNoteTextStyleNo 619 to assign TextStyleNo 581 property (in the constructor).

Limitation: in TRVReportHelper 503, all pages consisting only of endnotes' texts have the same height as the last document's page.

Use SetCurrentItemExtraIntPropertyEx 345 and SetCurrentItemExtraStrPropertyEx 346 to change properties of label items as editing operations.

Example

This example inserts a new endnote in rveMain (TCustomRichViewEdit). TextStyleNo 581 for this endnote is returned by RVGetNoteTextStyleNo 619 function basing on the rveMain.CurTextStyleNo 290-th style).

Document 586 for this endnote contains one reference 108 (TextStyleNo 581 for this reference is returned by RVGetNoteTextStyleNo 619 function basing on the 0th style) and one space character of the 0th style.

```
var EndNote: TRVEndnoteItemInfo;
    NoteRef: TRVNoteReferenceItemInfo;
begin
  EndNote := TRVEndnoteItemInfo.CreateEx(rveMain.RVData,
    RVGetNoteTextStyleNo(rvs, rveMain.CurTextStyleNo), 1, False);
  NoteRef := TRVNoteReferenceItemInfo.CreateEx(EndNote.Document, RVGetNoteTextStyleN
  EndNote.Document.AddItem('', NoteRef);
  EndNote.Document.AddNL(' ', 0, -1);
  if rveMain.InsertItem('', EndNote) then
    ...
end;
```

The topic about TRVNoteReferenceItemInfo 602 contains example showing how to edit Document 586.

See Also...

Demos:

- Demos\Delphi\Editors\Notes
- Demos\CBuilder\Editors\Notes

1.5.5.1 Methods

1.5.5.1.1 TRVEndnoteItemInfo.Create

A constructor, creates a new TRVEndnoteItemInfo object

```
constructor Create(RVData: TPersistent); override;
```

Parameter:

RVData – RVData 151 property of TRichView 128 or TRichViewEdit 282, where you wish to insert this item.

This constructor creates a new endnote with TextStyleNo 581=0.

There is more convenient constructor, CreateEx 589.

See also methods of TCustomRichView:

- AddItem 166.

See also methods of TCustomRichViewEdit:

- InsertItem 320.

1.5.5.1.2 TRVEndnoteItemInfo.CreateEx

A constructor, creates a new TRVEndnoteItemInfo object with the specified properties

```
constructor CreateEx(RVData: TPersistent;
  ATextStyleNo, AStartFrom: Integer; AReset: Boolean); override;
```

Parameters:

RVData – RVData 151 property of TRichView 128 or TRichViewEdit 282 where you wish to insert this item.

ATextStyleNo – value for TextStyleNo 581 property. It's recommended to use RVGetNoteTextStyleNo 619 function for assigning this parameter.

AStartFrom – value for StartFrom 585 property.

AReset – value for Reset 584 property.

See also methods:

- Create 588.

See also methods of TCustomRichView:

- AddItem 166.

See also methods of TCustomRichViewEdit:

- InsertItem 320.

1.5.6 Footnote - TRVFootnoteItemInfo

TRVFootnoteItemInfo is a class representing a footnote 104 in TRichView documents. This is not a component. Objects of this class are created at runtime and appended to TRichView (see AddItem 166) or inserted into TRichViewEdit (see InsertItem 320). Style 194 of this item type: *rvsFootnote* (-203)

Unit RVNote.

Syntax

```
TRVFootnoteItemInfo = class(TRVFootOrEndnoteItemInfo 586)
```

(introduced in version 10)

Hierarchy

TObject
TPersistent
TCustomRVItemInfo 526
TRVNonTextItemInfo
TRVRectItemInfo
TRVLabelItemInfo 579
TRVSeqItemInfo 582
TCustomRVNoteItemInfo 586
TRVFootOrEndnoteItemInfo 586

Using

Numbering type for footnotes is defined in FootnoteNumbering 389 property of TRVStyle 385 component (NumberType 584 property, inherited from TRVSeqItemInfo 582, is ignored).

You can enumerate all footnotes in documents using RVGetFirstFootnote and RVGetNextFootnote[618] functions.

Use RVGetNoteTextStyleNo[619] to assign TextStyleNo[581] property (in the constructor).

Footnotes are numbered continuously in TCustomRichView. But when printing, footnote numbering can be restarted on each page, see FootnotePageReset[392] property of TRVStyle[385] component.

Limitation: footnote's Document[586] must not be higher than the page height. Unlike endnotes[103], footnote is printed on a single page.

Use SetCurrentItemExtraIntPropertyEx[345] and SetCurrentItemExtraStrPropertyEx[346] to change properties of label items as editing operations.

Example

This example inserts a new footnote in rveMain (TCustomRichViewEdit). TextStyleNo[581] for this footnote is returned by RVGetNoteTextStyleNo[619] function basing on the rveMain.CurTextStyleNo[290]-th style).

Document[586] for this footnote contains one reference[108] (TextStyleNo[581] for this reference is returned by RVGetNoteTextStyleNo[619] function basing on the 0th style) and one space character of the 0th style.

```
var FootNote: TRVFootnoteItemInfo;
    NoteRef: TRVNoteReferenceItemInfo[602];
begin
  FootNote := TRVFootnoteItemInfo.CreateEx[591](rveMain.RVData[151],
    RVGetNoteTextStyleNo[619](rvs, rveMain.CurTextStyleNo[290]), 1, False);
  NoteRef := TRVNoteReferenceItemInfo[602].CreateEx[604](FootNote.Document[586], RVGetNoteTextStyle
  FootNote.Document[586].AddItem[166]('', NoteRef);
  FootNote.Document[586].AddNL[167](' ', 0, -1);
  if rveMain.InsertItem[320]('', FootNote) then
    ...
end;
```

The topic about TRVNoteReferenceItemInfo[602] contains example showing how to edit Document[586].

See Also...

Demos:

- Demos\Delphi\Editors\Notes
- Demos\CBuilder\Editors\Notes

1.5.6.1 Methods

1.5.6.1.1 TRVFootnoteItemInfo.Create

A constructor, creates a new TRVFootnoteItemInfo object

```
constructor Create(RVData: TPersistent); override;
```

Parameter:

RVData – RVData[151] property of TRichView[128] or TRichViewEdit[282], where you wish to insert this item.

This constructor creates a new footnote with TextStyleNo[581]=0.

There is more convenient constructor, CreateEx[591].

See also methods of TCustomRichView:

- AddItem[166].

See also methods of TCustomRichViewEdit:

- InsertItem[320].

1.5.6.1.2 TRVFootnoteItemInfo.CreateEx

A constructor, creates a new TRVFootnoteItemInfo object with the specified properties

```
constructor CreateEx(RVData: TPersistent;
  ATextStyleNo, AStartFrom: Integer; AReset: Boolean); override;
```

Parameters:

RVData – RVData 151 property of TRichView 128 or TRichViewEdit 282 where you wish to insert this item.

ATextStyleNo – value for TextStyleNo 581 property. It's recommended to use RVGetNoteTextStyleNo 619 function for assigning this parameter.

AStartFrom – value for StartFrom 585 property.

AReset – value for Reset 584 property.

See also methods:

- Create 590.

See also methods of TCustomRichView:

- AddItem 166.

See also methods of TCustomRichViewEdit:

- InsertItem 320.

1.5.7 Sidenote - TRVSidenoteItemInfo

TRVSidenoteItemInfo is a class representing a sidenote 105 in TRichView documents. Sidenote is a note displayed in a floating box.

This is not a component. Objects of this class are created at runtime and appended to TRichView (see AddItem 166) or inserted into TRichViewEdit (see InsertItem 320). Style 194 of this item type: *rvsSidenote* (-206)

Unit RVSidenote.

Syntax

```
TRVSidenoteItemInfo = class(TCustomRVNoteItemInfo 586)
```

(introduced in version 15)

Hierarchy

TObject
TPersistent
TCustomRVItemInfo 526
TRVNonTextItemInfo
TRVRectItemInfo
TRVLabelItemInfo 579
TRVSeqItemInfo 582
TCustomRVNoteItemInfo 586

Using

Numbering type for sidenotes is defined in SidenoteNumbering 389 property of TRVStyle 385 component (NumberType 584 property, inherited from TRVSeqItemInfo 582, is ignored).

You can enumerate all sidenotes in documents using RVGetFirstSidenote and RVGetNextSidenote 618 functions (these functions return text box items 107 as well)

Use RVGetNoteTextStyleNo 619 to assign TextStyleNo 581 property (in the constructor).

This class introduces the following properties:

- BoxPosition 592 – contains properties defining the position of the floating box

- BoxProperties 592 – contains properties describing size, background and border of the floating box.

Use SetCurrentItemExtraIntPropertyEx 345 and SetCurrentItemExtraStrPropertyEx 346 to change properties of sidenotes as editing operations.

Example

This example inserts a new sidenote in rveMain (TCustomRichViewEdit). TextStyleNo 581 for this sidenote is returned by RVGetNoteTextStyleNo 619 function basing on the rveMain.CurTextStyleNo 290-th style).

Document 586 for this sidenote contains one reference 108 (TextStyleNo 581 for this reference is returned by RVGetNoteTextStyleNo 619 function basing on the 0th style) and one space character of the 0th style.

A text box for this sidenote has a yellow background and a double border, and positioned by 50 pixels below the sidenote character line's top side:

```
var SideNote: TRVSidenoteItemInfo;
    NoteRef: TRVNoteReferenceItemInfo;
begin
  SideNote := TRVSidenoteItemInfo.CreateEx(rveMain.RVData,
    RVGetNoteTextStyleNo(rvs, rveMain.CurTextStyleNo), 1, False);
  NoteRef := TRVNoteReferenceItemInfo.CreateEx(SideNote.Document, RVGetNoteTextStyle
  SideNote.Document.AddItem('', NoteRef);
  SideNote.Document.AddNL(' ', 0, -1);
  SideNote.BoxProperties.Background.Color := clYellow;
  SideNote.BoxProperties.Border.Style := rvbDouble;
  SideNote.BoxPosition.VerticalOffset := rveMain.Style.PixelsToUnits(50);
  if rveMain.InsertItem('', SideNote) then
    ...
end;
```

The topic about TRVNoteReferenceItemInfo 602 contains example showing how to edit Document 586.

See also

- TRVTexBoxItemInfo 601

1.5.7.1 Properties

1.5.7.1.1 TRVSidenoteItemInfo.BoxProperties

Properties of a floating box: size, border, background, vertical alignment of content.

```
property BoxProperties: TRVBoxProperties;
```

1.5.7.1.2 TRVSidenoteItemInfo.BoxPosition

Properties of a floating box: horizontal and vertical positions, z-order.

```
property BoxPosition: TRVBoxPosition;
```

1.5.7.2 Methods

1.5.7.2.1 TRVSidenoteItemInfo.Create

A constructor, creates a new TRVSidenoteItemInfo object

```
constructor Create(RVData: TPersistent); override;
```

Parameter:

RVData – RVData 151 property of TRichView 128 or TRichViewEdit 282, where you wish to insert this item.

This constructor creates a new sidenote with TextStyleNo 581=0.

There is more convenient constructor, CreateEx [593].

See also methods of TCustomRichView:
- AddItem [166].

See also methods of TCustomRichViewEdit:
- InsertItem [320].

1.5.7.2.2 TRVSidenoteItemInfo.CreateEx

A constructor, creates a new TRVSidenoteItemInfo object with the specified properties

```
constructor CreateEx(RVData: TPersistent;
  ATextStyleNo, AStartFrom: Integer; AReset: Boolean); override;
```

Parameters:

RVData – RVData [151] property of TRichView [128] or TRichViewEdit [282] where you wish to insert this item.

ATextStyleNo – value for TextStyleNo [581] property. It's recommended to use RVGetNoteTextStyleNo [619] function for assigning this parameter.

AStartFrom – value for StartFrom [585] property.

AReset – value for Reset [584] property.

See also methods:
- Create [592].

See also methods of TCustomRichView:
- AddItem [166].

See also methods of TCustomRichViewEdit:
- InsertItem [320].

1.5.7.3 Classes of Properties

1.5.7.3.1 TRVBoxProperties

TRVBoxProperties contains properties of a floating box: size, border, background, etc.

Unit RVFloatingBox.

Syntax

```
TRVBoxProperties = class (TPersistent)
```

Hierarchy

TObject
TPersistent

Using

This is a class of BoxProperties [592] property of sidenotes [105] and text boxes [107].

Properties:
- Background [594];
- Border [594];
- Height, HeightType [594];
- Width, WidthType [595];
- VAlign [595].

1.5.7.3.1.1 Properties

Defines background properties.

```
type
  TRVBoxBackgroundRect = class (TRVBackgroundRect[466])
property Background: TRVBoxBackgroundRect;
```

TRVBoxBackgroundRect changes the initial values of properties introduced in TRVBackgroundRect[466]:

- Color[467]=*clWindow*
- BorderOffsets[466]: Left=Right=10, Top=Bottom=5.

Unlike in TRVBackgroundRect, in TRVBoxBackgroundRect negative values of BorderOffsets are ignored (treated as zeros).

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx[345] and SetCurrentItemExtraStrPropertyEx[346] to change value of subproperties of this property as an editing operation.

Defines background properties.

```
type
  TRVBoxBorder = class (TRVBorder[467])
property Border: TRVBoxBorder;
```

TRVBoxBorder changes the initial values of properties introduced in TRVBorder[467]:

- Style[468]=*rvbSingle*
- BorderOffsets[467]: Left=Right=10, Top=Bottom=5.

Unlike in TRVBorder, in TRVBoxBorder negative values of BorderOffsets are ignored (treated as zeros).

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx[345] and SetCurrentItemExtraStrPropertyEx[346] to change values of subproperties of this property as an editing operation.

These properties define a height of a floating box.

```
type
  TRVBoxHeightType = (rvbhtAbsolute, rvbhtRelPage, rvbhtRelMainTextArea,
    rvbhtRelTopMargin, rvbhtRelBottomMargin, rvbhtAuto);
property HeightType: TRVBoxHeightType;
property Height: TRVFloatSize[633];
```

Value of HeightType	Meaning
rvbhtAbsolute	**Height** specifies the box height in Units[404] of TRVStyle (pixels or twips)
rvbhtRelPage	**Height** specifies the box height in (1/1000)% of the page height
rvbhtRelMainTextArea	**Height** specifies the box height in (1/1000)% of the main text area height
rvbhtRelTopMargin	**Height** specifies the box height in (1/1000)% of the top margin height
rvbhtRelBottomMargin	**Height** specifies the box height in (1/1000)% of the bottom margin height
rvbhtAuto	The box height is calculated based on the box content. **Height** property is ignored.

A main text area is an area on a page between margins, where the main document is printed.

Heights of the main text area, the top margin and the bottom margins are calculated after adjusting by the header height and the footer height (too long header may increase a height the top margin and decrease a height of the main text area; too long footer may increase a height the bottom margin and decrease a height of the main text area). Since headers and footers may be different on first/odd/even pages, these height may be different as well.

These properties define a height of a text box including contents, padding and border.

Unlike position properties, these values are always calculated relative to a page (and never calculated relative to a table cell).

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx 345 and SetCurrentItemExtraStrPropertyEx 346 to change values of these properties as an editing operation.

Example:

HeightType=*rvbhtRelPage*, **Height**=20000: the height of box is equal to 20% of the page height.

Default values:

- HeightType: *rvbhtAuto*
- Height: 100

Specifies a vertical alignment of a box content inside a box.

```
property VAlign: TTextLayout;
```

This property makes sense if HeightType 594<>*rvbhtAuto*.

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx 345 and SetCurrentItemExtraStrPropertyEx 346 to change value of this property as an editing operation.

Default value:

tlTop

These properties define a width of a floating box.

```
type
  TRVBoxWidthType = (rvbwtAbsolute, rvbwtRelPage, rvbwtRelMainTextArea,
    rvbwtRelLeftMargin, rvbwtRelRightMargin,
    rvbwtRelInnerMargin, rvbwtRelOuterMargin);
property WidthType: TRVBoxWidthType;
property Width: TRVFloatSize 633;
```

Value of WidthType	Meaning
rvbwtAbsolute	**Width** specifies the box width in Units 404 of TRVStyle (pixels or twips)
rvbwtRelPage	**Width** specifies the box width in (1/1000)% of the page width
rvbwtRelMainTextArea	**Width** specifies the box width in (1/1000)% of the main text area width
rvbwtRelLeftMargin	**Width** specifies the box width in (1/1000)% of the left margin width
rvbwtRelRightMargin	**Width** specifies the box width in (1/1000)% of the right margin width
rvbwtRelInnerMargin	**Width** specifies the box width in (1/1000)% of the inner margin width (i.e. left margin for odd pages, right margin for even pages)
rvbwtRelOuterMargin	**Width** specifies the box width in (1/1000)% of the outer margin width (i.e. right margin for odd pages, left margin for even pages)

A main text area is an area on a page between margins, where the main document is printed.

"Mirror margins" option affects calculation of margin widths on even pages.

For text boxes included in a header/footer that is used both for odd and even pages, a calculation is performed for odd pages (the same width is used for all pages).

These properties define a width of a text box including contents, padding and border.

Unlike position properties, these values are always calculated relative to a page (and never calculated relative to a table cell).

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx[345] and SetCurrentItemExtraStrPropertyEx[346] to change values of these properties as an editing operation.

Example:

WidthType=*rvbwtRelPage*, **Width**=20000: the width of box is equal to 20% of the page width.

Default values:

- WidthType: *rvbwtAbsolute*
- Height: 100

1.5.7.3.2 TRVBoxPosition

TRVBoxPosition contains properties defining a position of a floating box.

Unit RVFloatingPos.

Syntax

```
TRVBoxPosition = class (TPersistent)
```

Hierarchy

TObject
TPersistent

Using

This is a class of BoxPosition[592] property of sidenotes[105] and text boxes[107].

Properties:

- HorizontalAnchor[596] and HorizontalPositionKind, HorizontalOffset, HorizontalAlignment[597];
- VerticalAnchor[599] and VerticalPositionKind, VerticalOffset, VerticalAlignment[600];
- RelativeToCell[599];
- PositionInText[599].

1.5.7.3.2.1 Properties

Specifies a horizontal anchor area for a floating box.

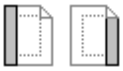
```
type
  TRVHorizontalAnchor = (rvhanCharacter, rvhanPage, rvhanMainTextArea,
    rvhanLeftMargin, rvhanRightMargin, rvhanInnerMargin, rvhanOuterMargin);
property HorizontalAnchor: TRVHorizontalAnchor;
```

A text box is positioned horizontally relative to the area specified in this property.

It is important to understand that each area has a width; for example, when printing using TRVPrint[479], a left margin starts from the page's left edge and ends at Margins[486].Left.

If RelativeToCell[599]=*True*, and a sidenote[105] (or a text box item[107]) is inserted in a table cell[113], then this property defines an area of this cell instead of a page.

Value	Anchor Area	Scheme
rvhanCharacter	A sidenote[105] character, or a place of insertion of a text box item[107].	
rvhanPage	A whole page (or a whole table cell)	

rvhanMainTextArea	A main text area, i.e. an area between margins (or a table cell area minus cellpadding)	
rvhanLeftMargin	A left page margin (or a left cell padding area)	
rvhanRightMargin	A right page margin (or a right cell padding area)	
rvhanInnerMargin	An inner page margin: a left margin for odd pages, a right margin for even pages (or a left cell padding area on odd pages, a right cell padding area on even pages)	
rvhanOuterMargin	An outer page margin: a right margin for odd pages, a left margin for even pages (or a right cell padding area on odd pages, a left cell padding area on even pages)	

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx[345] and SetCurrentItemExtraStrPropertyEx[346] to change value of this property as an editing operation.

Differences from Microsoft Word*:

- In MS Word, a "character" anchor does not have a width, it is just a position. In TRichView, sidenote character has a width, so a center and a right alignments relative to a character of a sidenote are slightly different in TRichView and in MS Word (for simple text boxes, they are identical).
- MS Word does not support a relative (percent) alignment relative to a character (so TRichView exports it as an absolute position with zero offset).

(* this information is based on test of MS Word 2013)

Default value:

rvhanCharacter

These properties define a horizontal position of a floating box.

```
type
  TRVFloatHorizontalAlignment = (rvfphalLeft, rvfphalCenter, rvfphalRight);
property HorizontalPositionKind: TRVFloatPositionKind[632];
property HorizontalOffset: TRVFloatPosition[632];
property HorizontalAlignment: TRVFloatHorizontalAlignment;
```

A position is calculated relative to the area specified in HorizontalAnchor[596].

Note 1: these properties define the position of the box in the way similar to Microsoft Word. However, there are differences, see the notes in the topic about HorizontalAnchor[596].

Note 2: printers cannot print at areas close to paper edges; is the box is positioned close to a left or a right side of the page, it may overlap these areas.

HorizontalPositionKind=rvfpkAlignment

The position is defined in **HorizontalAlignment**. **HorizontalOffset** is ignored.

Value of HorizontalAlignment	Meaning	Scheme

rvfphalLeft	The left side of the box is at the left side of the anchor area.	
rvfphalCenter	The center of the box is at center of the anchor area.	
rvfphalRight	The right side of the box is at the right side of the anchor area.	

HorizontalPositionKind=rvfpkAbsPosition

The position is defined in **HorizontalOffset**. **HorizontalAlignment** is ignored.

HorizontalOffset is measured in pixels or twips, depending on Units 404 property of the linked TRVStyle 385.

A position is counted from the left side of the anchor area. Positive values shift the box to the right, negative values shift the box to the left.

HorizontalPositionKind=rvfpkPercentPosition

The position is defined in **HorizontalOffset**. **HorizontalAlignment** is ignored.

HorizontalOffset is measured in (1/1000)% of the anchor area width.

A position is counted from the left side of the anchor area. Positive values shift the box to the right, negative values shift the box to the left.

Examples:

Value of HorizontalOffset	Meaning	Scheme
0	0%	
100000	100%	
50000	50%	
-100000	-100%	

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx 345 and SetCurrentItemExtraStrPropertyEx 346 to change values of these properties as an editing operation.

Default values:

- HorizontalPositionKind: *rvfpkAbsPosition*

- HorizontalOffset: 0
- HorizontalAlignment: *rvfphalLeft*

Specifies how the text box is positioned relative to a text where a sidenote [105] (or a text box item [107]) is inserted.

```
type
  TRVFloatPositionInText = (rvpitAboveText, rvpitBelowText);
property PositionInText: TRVFloatPositionInText;
```

Value	Meaning
rvpitAboveText	The box is above a text.
rvpitBelowText	The box is below a text.

Note: boxes inserted in a header or a footer are always below the main text (and below boxes inserted in the main text).

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx [345] and SetCurrentItemExtraStrPropertyEx [346] to change value of this property as an editing operation.

Default value:

rvpitAboveText

Allows anchoring a text box to a table cell [113].

```
property RelativeToCell: Boolean;
```

If *True*, and a sidenote [105] (or a text box item [107]) is placed in a table cell, then horizontal [596] and vertical [599] positions are calculated relative to this cell (instead of a page).

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx [345] and SetCurrentItemExtraStrPropertyEx [346] to change value of this property as an editing operation.

Default value:

True

Specifies a vertical anchor area for a floating box.

```
type
  TRVVerticalAnchor = (rvvanLine, rvvanParagraph, rvvanPage,
    rvvanMainTextArea, rvvanTopMargin, rvvanBottomMargin);
property VerticalAnchor: TRVVerticalAnchor;
```

A text box is positioned vertically relative to the area specified in this property.

It is important to understand that each area has a height; for example, when printing using TRVPrint [479], a top margin starts from the page's top edge and ends at Margins [486].Top.

If RelativeToCell [599]=*True*, and a sidenote [105] (or a text box item [107]) is inserted in a table cell [113], then this property defines an area of this cell instead of a page.

Value	Anchor Area	Scheme
rvvanLine	A line where a sidenote [105] character (or a text box item [107]) is inserted.	
rvvanParagraph	A paragraph where a sidenote [105] character (or a text box item [107]) is inserted.	

rvvanPage	A whole page (or a whole table cell)	
rvvanMainTextArea	A main text area, i.e. an area between margins (or a table cell area minus cellpadding)	
rvvanTopMargin	A top page margin (or a top cell padding area)	
rvvanBottomMargin	A bottom page margin (or a bottom cell padding area)	

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx[345] and SetCurrentItemExtraStrPropertyEx[346] to change value of this property as an editing operation.

Differences from Microsoft Word*:

- In MS Word, a "line" anchor does not have a height. MS Word uses only a top line coordinate, so a center and a bottom alignments relative to a line are different in TRichView and MS Word.
- MS Word does not support alignment relative to a paragraph (so TRichView exports it as an alignment to a line).
- MS Word does not support a relative (percent) alignment relative to a line or a paragraph (so TRichView exports them as absolute positions with zero offsets).
- MS Word does not implement all options for vertical positioning relative to a cell:
  - absolute positions for all options (except for a line and a paragraph) are calculated relative to the cell area minus cellpadding (i.e. top of cell + top cell padding).
  - any alignment or percent position moves the box exactly to the cell area minus cellpadding (i.e. top of cell + top cell padding).

(* this information is based on tests of MS Word 2013)

Default value:

rvvanLine

These properties define a vertical position of a floating box.

```
type
  TRVFloatVerticalAlignment = (rvfpvalTop, rvfpvalCenter, rvfpvalBottom);
property VerticalPositionKind: TRVFloatPositionKind[632];
property VerticalOffset: TRVFloatPosition[632];
property VerticalAlignment: TRVFloatVerticalAlignment;
```

A position is calculated relative to the area specified in VerticalAnchor[599].

Note 1: these properties define the position of the box in the way similar to Microsoft Word. However, there are differences, see the notes in the topic about VerticalAnchor[599].

Note 2: printers cannot print at areas close to paper edges; is the box is positioned close to a top or a bottom side of the page, it may overlap these areas.

VerticalPositionKind=rvfpkAlignment

The position is defined in **VerticalAlignment**. **VerticalOffset** is ignored.

Value of VerticalAlignment	Meaning
rvfphalTop	The top side of the box is at the top side of the anchor area.

rvfphalCenter	The center of the box is at center of the anchor area.
rvfphalBottom	The bottom side of the box is at the bottom side of the anchor area.

VerticalPositionKind=rvfpkAbsPosition

The position is defined in **VerticalOffset**. **VerticalAlignment** is ignored.

VerticalOffset is measured in pixels or twips, depending on Units 404 property of the linked TRVStyle 385.

A position is counted from the top side of the anchor area. Positive values shift the box down, negative values shift the box up.

VerticalPositionKind=rvfpkPercentPosition

The position is defined in **VerticalOffset**. **VerticalAlignment** is ignored.

VerticalOffset is measured in (1/1000)% of the anchor area height.

A position is counted from the top side of the anchor area. Positive values shift the box down, negative values shift the box up.

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx 345 and SetCurrentItemExtraStrPropertyEx 346 to change values of these properties as an editing operation.

Default values:

- VerticalPositionKind: *rvfpkAbsPosition*
- VerticalOffset: 0
- VerticalAlignment: *rvfpvalTop*

1.5.8 TextBox - TRVTextBoxItemInfo

TRVTextBoxItemInfo is a class representing text box item 107 in TRichView documents. This item is printed as a floating box.

This is not a component. Objects of this class are created at runtime and appended to TRichView (see AddItem 166) or inserted into TRichViewEdit (see InsertItem 320). Style 194 of this item type: *rvsTextBox* (-207)

Unit RVSidenote.

Syntax

```
TRVTextBoxItemInfo = class(TRVSidenoteItemInfo 591)
```

(introduced in version 15)

Hierarchy

TObject
TPersistent
TCustomRVItemInfo 526
TRVNonTextItemInfo
TRVRectItemInfo
TRVLabelItemInfo 579
TRVSeqItemInfo 582
TCustomRVNoteItemInfo 586
TRVSidenoteItemInfo 591

Using

This is a simplified version of a sidenote 105. It has all properties of a sidenote, but the most of them are ignored,

except for BoxPosition[592] and BoxProperties[592].

A special mark can be drawn in places of insertion of text box items, see TRVStyle.FloatingLineColor[390].

1.5.8.1 Methods

1.5.8.1.1 TRVTextBoxItemInfo.Create

A constructor, creates a new TRVFootnoteItemInfo object

```
constructor Create(RVData: TPersistent); override;
```

Parameter:

RVData – RVData[151] property of TRichView[128] or TRichViewEdit[282], where you wish to insert this item.

This constructor creates a new text box item with TextStyleNo[581]=0.

See also CreateEx[602].

See also methods of TCustomRichView:

- AddItem[166].

See also methods of TCustomRichViewEdit:

- InsertItem[320].

1.5.8.1.2 TRVTextBoxItemInfo.CreateEx

A constructor, creates a new TRVTextBoxItemInfo object.

```
constructor CreateEx(RVData: TPersistent;
  ATextStyleNo, AStartFrom: Integer; AReset: Boolean); override;
```

The most of parameters of this constructor are not used. You can use a simpler constructor: Create[602].

Parameters:

RVData – RVData[151] property of TRichView[128] or TRichViewEdit[282] where you wish to insert this item.

ATextStyleNo – ignored.

AStartFrom – value for StartFrom[585] property. Assigned, but not used in this type of item.

AReset – value for Reset[584] property. Assigned, but not used in this type of item.

See also methods:

- Create[602].

See also methods of TCustomRichView:

- AddItem[166].

See also methods of TCustomRichViewEdit:

- InsertItem[320].

1.5.9 Reference to the Parent Note - TRVNoteReferenceItemInfo

TRVNoteReferenceItemInfo is an item type referring to[108] the parent footnote[104], endnote[103], or sidenote[105] item. Objects of this class are usually appended to the note's Document[586] (see AddItem[166]). Besides, they can be appended to TRichView (see AddItem[166]) or inserted into TRichViewEdit (see InsertItem[320]). Style[194] of this item type: *rvsNoteReference* (-205).

Unit RVNote.

Syntax

```
TRVNoteReferenceItemInfo = class(TRVLabelItemInfo[579])
```

(introduced in version 10)

Hierarchy

TObject
TPersistent
TCustomRVItemInfo 526
TRVNonTextItemInfo
TRVRectItemInfo
TRVLabelItemInfo 579

Using

When inserted in a note's Document 586, the note reference displays the same text as the parent note.

When inserted in TCustomRichView, it displays NoteText 146.

Text 581 for this item is calculated automatically.

Use RVGetNoteTextStyleNo 619 to assign TextStyleNo 581 property (in the constructor).

Use SetCurrentItemExtraIntPropertyEx 345 and SetCurrentItemExtraStrPropertyEx 346 to change properties of label items as editing operations.

Example

See examples in TRVFootnoteItemInfo 589 and TRVEndNoteItemInfo 587 showing how to insert footnote/endnote with document containing note reference.

This example shows how to edit note document in rveNote (TCustomRichViewEdit 282). The footnote/endnote is NoteItem.

This code copies document from footnote/endnote to the editor:

```
var Stream: TMemoryStream;
begin

  rveNote.NoteText 146 := NoteItem.Text 581;
  rveNote.Clear 175;
  Stream := TMemoryStream.Create;
  try
    NoteItem.Document 586.SaveRVFToStream 220(Stream);
    Stream.Position := 0;
    rveNote.LoadRVFFromStream 208(Stream);
  finally
    Stream.Free;
  end;
  with rveNote do begin
    Format 182;
    SetSelectionBounds 235(ItemCount-1, GetOffsAfterItem(ItemCount-1),
      ItemCount-1, GetOffsAfterItem(ItemCount-1));
  end;
```

You can see that the text of footnote/endnote is assigned to the editor's NoteText 146. If this document contains note references, they will display this text.

After editing, the document can be saved back to the note:

```
  if not rveNote.Modified 293 then
    exit;
  Stream := TMemoryStream.Create;
  try
    rveNote.SaveRVFToStream 220(Stream, False);
    NoteItem.ReplaceDocumentEd 587(Stream);
```

```
  finally
    Stream.Free;
  end;
  rveNote.Modified[293] := False;
```

See Also...

Demos:

- Demos\Delphi\Editors\Notes
- Demos\CBuilder\Editors\Notes

1.5.9.1 Methods

1.5.9.1.1 TRVNoteReferenceItemInfo.Create

A constructor creating a new TRVNoteReferenceItemInfo object

```
constructor Create(RVData: TPersistent); override;
```

Parameter:

RVData – RVData[151] property of TRichView[128] or TRichViewEdit[282], where you wish to insert this item.

This constructor creates a new note reference with TextStyleNo[581]=0.

There is more convenient constructor, CreateEx[604].

See also methods of TCustomRichView:

- AddItem[166].

See also methods of TCustomRichViewEdit:

- InsertItem[320].

1.5.9.1.2 TRVNoteReferenceItemInfo.CreateEx

A constructor creating a new TRVNoteReferenceItemInfo object with the specified properties.

```
constructor CreateEx(RVData: TPersistent; TextStyleNo: Integer);
```

Parameters:

RVData – RVData[151] property of TRichView[128] or TRichViewEdit[282] where you wish to insert this item.

TextStyleNo – value for TextStyleNo[581] property. It's recommended to use RVGetNoteTextStyleNo[619] function for assigning this parameter.

See also methods:

- Create[604].

See also methods of TCustomRichView:

- AddItem[166].

See also methods of TCustomRichViewEdit:

- InsertItem[320].

1.5.10 Ancestor for Text Fields - TCustomRVFieldItemInfo

TCustomRVFieldItemInfo is an ancestor class for page number and page count fields.

Objects of this class must not be created directly, create objects of the following inherited classes:

- TRVPageNumberItemInfo[605];
- TRVPageCountItemInfo[607].

Unit RVFieldItems.

Syntax

```
TCustomRVFieldItemInfo = class (TRVLabelItemInfo[579]);
TCustomRVPageNumberItemInfo = class(TCustomRVFieldItemInfo);
```

(introduced in version 15)

Hierarchy

TObject
TPersistent
TCustomRVItemInfo[526]
TRVNonTextItemInfo
TRVRectItemInfo
TRVLabelItemInfo[579]

TCustomRVPageNumberItemInfo introduces a new property: NumberType[605].

1.5.10.1 Properties

1.5.10.1.1 TCustomRVPageNumberItemInfo.NumberType

Specifies how the page number (or page count) is displayed.

```
NumberType: TRVPageNumberType[637];
```

This property is initialized in the constructor.

Use TRichViewEdit.SetCurrentItemExtraIntPropertyEx[345] to change value of this property as an editing operation.

Default value:

rvpntDefault

1.5.11 Page Number - TRVPageNumberItemInfo

TRVPageNumberItemInfo displays the current page number[109].

Unit RVFieldItems.

Syntax

```
TRVPageNumberItemInfo = class(TCustomRVPageNumberItemInfo[604])
```

(introduced in version 15)

Hierarchy

TObject
TPersistent
TCustomRVItemInfo[526]
TRVNonTextItemInfo
TRVRectItemInfo
TRVLabelItemInfo[579]

Using

When inserted in TRichView[128], it is displayed as Text[581] (by default, as '{p}').

When printed with TRVPrint[479], it displays a number of the page where this item is located. Page numbers are counted from TRVPrint.PageNoFromNumber[516].

When drawing with TRVReportHelper[503], the default behavior is like in TRVPrint, but you can provide a value for the page number in OnGetPageNumber[508] event.

ScaleRichView: when inserted in TSRichViewEdit, it displays a number of the page where this item is located. Page numbers are counted from TSRichViewEdit.PageProperty.PageNoFromNumber.

A displayed page number is formatted according to NumberType[605] property.

1.5.11.1 Methods

1.5.11.1.1 TRVPageNumberItemInfo.Create

A constructor creating a new TRVPageNumberItemInfo object

```
constructor Create(RVData: TPersistent); override;
```

Parameter:

RVData – RVData[151] property of TRichView[128] or TRichViewEdit[282], where you wish to insert this item.

This constructor creates a new page number field with TextStyleNo[581]=0, Text[581]='{p}'.

There is more convenient constructor, CreateEx[606].

See also methods of TCustomRichView:

- AddItem[166].

See also methods of TCustomRichViewEdit:

- InsertItem[320].

1.5.11.1.2 TRVPageNumberItemInfo.CreateEx

A constructor creating a new TRVPageNumberItemInfo object with the specified properties.

```
constructor CreateEx(RVData: TPersistent; NumberType: TRVPageNumberType[637];
  TextStyleNo: Integer); reintroduce;
```

Parameters:

RVData – RVData[151] property of TRichView[128] or TRichViewEdit[282] where you wish to insert this item.

NumberType – value for NumberType[605] property.

TextStyleNo – value for TextStyleNo[581] property.

This constructor creates a new page number field with Text[581]='{p}'.

See also methods:

- Create[606].

See also methods of TCustomRichView:

- AddItem[166].

See also methods of TCustomRichViewEdit:

- InsertItem[320].

1.5.12 Page Count - TRVPageCountItemInfo

TRVPageCountItemInfo displays the count of pages[110].

Unit RVFieldItems.

Syntax

```
TRVPageCountItemInfo = class(TCustomRVPageNumberItemInfo[604])
```

(introduced in version 15)

Hierarchy

TObject
TPersistent
TCustomRVItemInfo[526]
TRVNonTextItemInfo
TRVRectItemInfo
TRVLabelItemInfo[579]

Using

When inserted in TRichView[128], it is displayed as Text[581] (by default, as '{P}').

When printed with TRVPrint[479], it displays a count of pages.

ScaleRichView: when inserted in TSRichViewEdit, it displays a count of pages.

A displayed page number is formatted according to NumberType[605] property.

1.5.12.1 Methods

1.5.12.1.1 TRVPageCountItemInfo.Create

A constructor creating a new TRVPageCountItemInfo object

```
constructor Create(RVData: TPersistent); override;
```

Parameter:

RVData – RVData[151] property of TRichView[128] or TRichViewEdit[282], where you wish to insert this item.

This constructor creates a new page count field with TextStyleNo[581]=0, Text[581]='{P}'.

There is more convenient constructor, CreateEx[607].

See also methods of TCustomRichView:

- AddItem[166].

See also methods of TCustomRichViewEdit:

- InsertItem[320].

1.5.12.1.2 TRVPageCountItemInfo.CreateEx

A constructor creating a new TRVPageCountItemInfo object with the specified properties.

```
constructor CreateEx(RVData: TPersistent; NumberType: TRVPageNumberType[637];
  TextStyleNo: Integer); reintroduce;
```

Parameters:

RVData – RVData[151] property of TRichView[128] or TRichViewEdit[282] where you wish to insert this item.

NumberType – value for NumberType[605] property.

TextStyleNo – value for TextStyleNo[581] property.

This constructor creates a new page count field with Text[581]='{P}'.

See also methods:

- Create[607].

See also methods of TCustomRichView:

- AddItem[166].

See also methods of TCustomRichViewEdit:

- InsertItem[320].

1.6 Other Classes

1.6.1 ERichViewError

This is an exception class used by TRichView Family of components. It does not introduce new methods and properties.

Unit RichView;

Syntax

```
ERichViewError = class(Exception)
```

Hierarchy

TObject
Exception

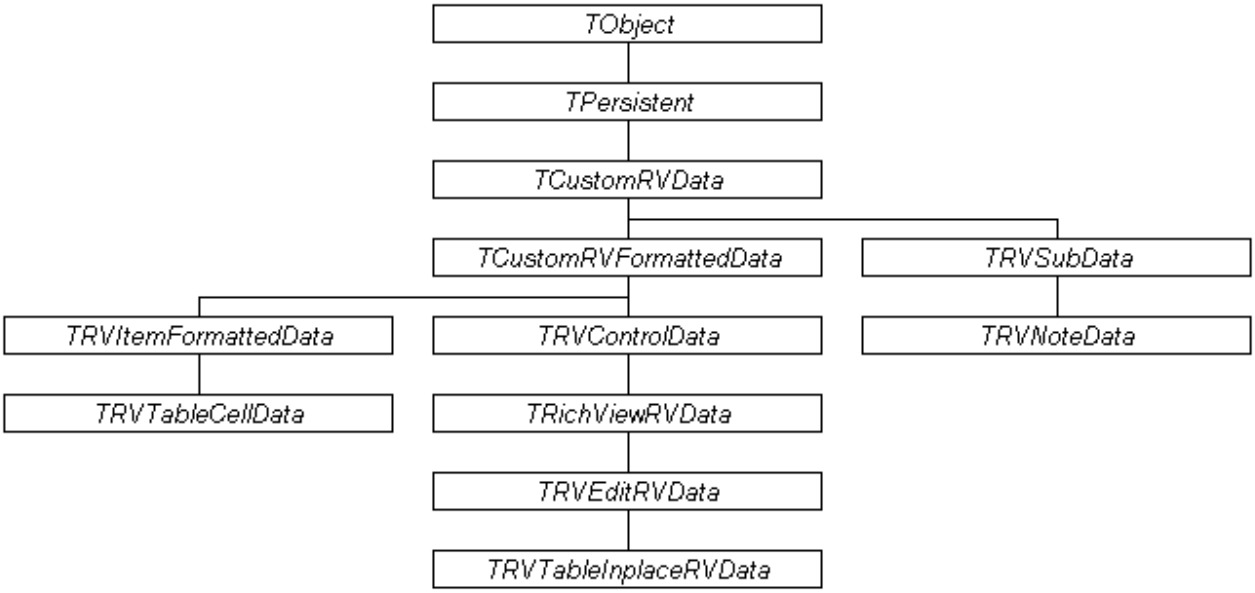
1.6.2 TCustomRVData and Others (TRichView Documents)

These classes represent RichView documents. Documents contain a list of items. Some items may, in each order, contain subdocuments.

Class	Unit	Meaning
TCustomRVData	CRVData	Basic class for TRichView documents.
TCustomRVFormattedData	CRVFData	Basic class for formatted TRichView documents (allows storing items coordinates).
TRVControlData	RVCtrlData	Basic class for documents owned by windowed control.
TRichViewRVData	RVRVData	Document in TRichView[128] and TDBRichView[373], a class of RVData[151] property of these components.
TRVEditRVData	RVERVData	Document in TRichViewEdit[282] and TDBRichViewEdit[377], a class of RVData[151] property of these components.
TRVTInplaceRVData	RVTInplace	Document in cell inplace editor, a class of of RVData[151] property of cell inplace editors.
TRVItemFormattedData	RVDataList	Basic class for formatted subdocuments owned by items.
TRVTableCellData[567]	RVTable	Table[111] cell.
TRVSubData	RVSubData	Basic class for invisible documents owned by items.
TRVNoteData	RVNote	Document in footnote or endnote.

This list does not include document classes used for printing.

Hierarchy



Overview

Special methods for **TCustomRVData**:

- Edit [610];
- GetRVData [611];
- GetSourceRVData [611].

Special methods for **TCustomRVFormattedData**:

- GetOriginEx [611].

These classes have many methods and properties similar to methods and properties in TRichView (what's why in this manual many links to RVData's methods refer to TRichView's methods).

For example, the methods for appending items:

(the links are to the corresponding methods of TRichView)

- AddBreak, AddBreakEx, AddBreakExTag, AddBreakTag, [160],
- AddBulletEx, AddBulletExTag [160],
- AddControlEx, AddControlExTag [162],
- AddHotspotEx, AddHotspotExTag [164],
- AddFmt [163], AddItem [166],
- AddNL, AddNLTag [167], AddTextNL [171],
- AddPictureEx, AddPictureExTag [169],
- AddNLATag, AddNLWTag [167]

Obtaining information from cell:

- GetItemStyle [194], GetItemText -A -W [195],
- GetBreakInfo [183], GetBulletInfo [184], GetControlInfo [187], GetHotspotInfo [189], GetPictureInfo [201]
- GetItemTag [195], GetItemVAlign [196], IsParaStart [208], GetItemPara [194], IsFromNewLine [207]

Modifying items:

- SetBreakInfo [226], SetBulletInfo [227], SetControlInfo [228], SetHotspotInfo [229], SetPictureInfo [234], SetItemTag [231], SetItemVAlign [233], SetItemText -A -W [232]

See also

- Controls, Documents, Items 72

1.6.2.1 TCustomRVData.Edit

Initializes editor for this RVData.

```
function Edit: TCustomRVData;
```

For RVData 151 of editor (main editor, or cell inplace editor), this method does nothing.

For table cell 567, this method creates cell inplace editor and returns its RVData 151.

If you want to select part of document in table cell, call Cell.Edit (or table.EditCell 552) before calling cell.SetSelectionBounds 235 or cell.SelectAll 223.

This method must be called when the document is formatted.

Return value

GetRVData 611 of the given RVData after initialization of editing.

Example

This example deletes all controls from rve: TRichViewEdit 282 as an editing operation (that can be undone and redone by the user).

In this example, all references to the methods and properties of TCustomRVFormattedData point to the methods and properties of TRichView 128. TCustomRVFormattedData has all these methods and properties, with the same meaning.

```
procedure DeleteAllControls(RVData: TCustomRVData608; rve: TCustomRichViewEdit282);
var i,r,c: Integer;
    table: TRVTableItemInfo528;
begin
  i := 0;
  while i<RVData.ItemCount144 do begin
    case RVData.GetItemStyle194(i) of
    rvsComponent663:
      begin
        RVData := RVData.Edit;
        TCustomRVFormattedData(RVData).SetSelectionBounds235(i, 0, i, 1);
        rve.DeleteSelection306;
        i := rve.TopLevelEditor295.CurItemNo288-1;
      end;
    rvsTable663:
      begin
        table := TRVTableItemInfo528(RVData.GetItem190(i));
        for r := 0 to table.RowCount543-1 do
          for c := 0 to table.ColCount538-1 do
            if table.Cells537[r,c]<>nil then
              DeleteAllControls(table.Cells537[r,c].GetRVData575, rve);
      end;
    end; // of case
    inc(i);
  end; // of while
end;
```

Call:

```
DeleteAllControls(rve.RVData151, rve);
```

1.6.2.2 TCustomRVData.GetRVData

For the given document, returns the document containing its items.

```
function GetRVData: TCustomRVData;
```

For the most of document classes, this method returns Self (RVData.GetRVData = RVData).

The only exception is table cells (TRVTableCellData[567]). When the cell is not edited, Cell.GetRVData = Cell. But when the cell is edited, this method returns RVData[151] of the cell inplace editor.

Using RVData.GetRVData instead of RVData you can always access the document's items.

See also

- GetSourceRVData[611];
- Edit[610].

1.6.2.3 TCustomRVData.GetSourceRVData

This is an inverse method to GetRVData[611].

```
function GetSourceRVData: TCustomRVData;
```

For the most of document classes, this method returns Self, i.e. RVData.GetSourceRVData = RVData.

The only exception is RVData[151] of the cell inplace editor, it returns the source cell (TRVTableCellData[567]) for this editor.

See also

- GetRVData[611].

1.6.2.4 TCustomRVFormattedData.GetOriginEx

Returns coordinates of the given subdocument relative to the main document.

```
procedure GetOriginEx(var ALeft, ATop: Integer);
```

Output parameters

ALeft, **ATop** – coordinates of the top left corner of this RVData relative to the top left corner of the main document.

Coordinates of the given item relative to the main document are calculated as a sum of values returned by RVData.GetItemCoords[191] and RVData.GetOriginEx.

This method does not take cell rotation into account, the coordinates are returned as if none of cells are rotated.

1.6.3 TPrintableRV

TRichView Reference[26] | Classes

TPrintableRV is used internally by TRVPrint

1.6.4 TRVBooleanRect

This class contains 4 boolean properties: Left, Top, Right, Bottom

Unit RVStyle;

Syntax

```
TRVRect = class (TPersistent)
```

(Introduced in version 1.2)

This class is used for defining visible sides for paragraph, cell and table borders.

See also:

- TRVBorder.VisibleBorders[468];
- TRVTableCellData.VisibleBorders[575];
- TRVTableItemInfo.VisibleBorders[544].

1.6.5 TRVDeleteUnusedStylesData

This class allows to delete unused text[404], paragraph[398] and list[396] styles in multiple documents (to delete styles that do not present in at least one of the documents)

Unit RVItem.

Syntax

```
TRVDeleteUnusedStylesData = class;
```

(introduced in version 10)

The only method of this class you need to use is the constructor:

```
constructor Create(ATextStyles, AParaStyles, AListStyles: Boolean);
```

Its parameters specify types of unused styles you want to delete.

The examples below assume that collections of text, paragraph and list styles are not saved in RVF documents[59] (but stored in file or in the Registry). In this case, several documents share the same collections of styles. TCustomRichView.DeleteUnusedStyles[181] cannot be used in this case, MarkStylesInUse[213] and DeleteMarkedStyles[180] must be used instead.

Note: usually it's much more simple (and recommended) to use one RVStyle per one RichView and store collections of styles inside RVF documents.

Example 1

How to remove unused styles in RichView1 and RichView2, linked to the same RVStyle.

```
uses RVItem;

  data := TRVDeleteUnusedStylesData.Create(True, True, True);
  RichView1.MarkStylesInUse(data);
  RichView2.MarkStylesInUse(data);
  RichView1.DeleteMarkedStyles(data);
  RichView2.DeleteMarkedStyles(data);
  data.Free;
```

Example 2

How to remove unused styles in several documents.

```
uses CRVData, RVItem;

  data := TRVDeleteUnusedStylesData.Create(True, True, True);
  RichView1.LoadRVF('Doc1.rvf');
  RichView1.MarkStylesInUse(data);
  RichView1.LoadRVF('Doc2.rvf');
  RichView1.MarkStylesInUse(data);
  ...
  RichViewDoNotCheckRVFStyleRefs := True;
  RichView1.LoadRVF('Doc1.rvf');
  RichView1.DeleteMarkedStyles(data);
  RichView1.SaveRVF('Doc1.rvf', False);
```

```
  RichView1.LoadRVF[211]('Doc2.rvf');
  RichView1.DeleteMarkedStyles[180](data);
  RichView1.SaveRVF[220]('Doc2.rvf', False);
  ...
  RichViewDoNotCheckRVFStyleRefs[653] := False;
  data.Free;
```

1.6.6 TRVGraphicHandler

This class provides an abstraction layer for managing graphic in TRichView components. It is designed to make TRichView components independent of specific graphic classes.

Unit RVGrHandler;

Syntax

```
TRVGraphicHandler = class;
```

(introduced in version 15)

Using

A single variable of this class is created: RVGraphicHandler[662]. You can call methods of the object returned in this variable.

Additionally, you can create a class inherited from TRVGraphicHandler and assign it to this variable (however, this possibility is not discussed in this help file).

Methods

A complete description of all methods of this class is beyond the scope of this help file.

The following methods are useful:

```
type
  TRVGraphicType = // defined in RVFuncs unit
    (rvgtBitmap, rvgtIcon, rvgtMetafile, rvgtJPEG, rvgtPNG, rvgtGIF, rvgtTIFF, rvgtOther)
procedure SetDefaultGraphicClass(GraphicType: TRVGraphicType; GraphicClass: TGraphicClass
procedure RegisterPngGraphic(GraphicClass: TGraphicClass);
procedure RegisterJpegGraphic(GraphicClass: TGraphicClass);
procedure RegisterHTMLGraphicFormat(GraphicClass: TGraphicClass);
function CreateGraphic(GraphicClass: TGraphicClass): TGraphic;
function CreateGraphicByType(GraphicType: TRVGraphicType): TGraphic;
function LoadFromFile(const FileName: String): TGraphic;
function LoadFromStream(Stream: TStream): TGraphic;
```

Choosing default graphic classes

RegisterPngGraphic registers a PNG graphic class for using in TRichView. Call this method one time when the application starts (for example, in the initialization section of the main form's unit). If PNG class is registered, TRichView controls can save and load PNG images in RTF and and save in DocX files. Optionally, bitmaps can be saved as PNG in RTF too, see TCustomRichView.RTFOptions[150] property (*rvrtfPNGInsteadOfBitmap*).

By default, in Delphi 2009+, TRichView registers a standard TPngImage as PNG graphic format, so, unless you want to use another class for PNG images, this method is necessary only for Delphi 2007 or older. For example, you can use Gustavo Huffenbacher Daud's TPngObject: ***download file*** (free PNG graphic class).

RegisterJpegGraphic registers a JPEG graphic class for using in TRichView. If this method is not called, TRichView uses TJPEGImage. This class is used when loading Jpeg images embedded in RTF.

There is an universal method allowing to define graphic classes for the specified graphic type: **SetDefaultGraphicClass**.

By default, the classes are:

- bitmaps: TBitmap,
- icons: TIcon,
- metafiles: TMetafile,
- jpeg: TJPEGImage,
- png: TPngImage (for Delphi 2009+, undefined otherwise)
- gif: TGifImage (for Delphi 2007+, if RVGifAnimate2007 is included in the project; for previous versions of Delphi, if RVGifAnimate is included in the project; undefined otherwise
- tiff: TWicImage (for Delphi 2010+, undefined otherwise)

HTML export

RegisterHTMLGraphicFormat allows to define HTML graphic formats. Call this method one time when the application starts (for example, in the initialization section of the main form's unit). Images of these formats are saved in HTML [62] as they are, without conversion to Jpegs. It's not necessary to call it for TJpegImage. These formats must have file extension associated, see TPicture.RegisterFileFormat. TRichView registers the following classes as HTML graphic formats:

- standard TPngImage (for Delphi 2009+);
- standard TGifImage (for Delphi 2007+), if you include RVGifAnimate2007 unit in your project;
- Anders Melander's TGifImage, if you include RVGifAnimate unit in your project;
- TJvGifImage from JEDI's JVCL, if you include RVJvGifAnimate unit in your project.

Creating graphics

CreateGraphic creates a graphic of this specified class. For Delphi/C++Builder 6 or newer, this method is equivalent to calling GraphicClass.Create. For older version of Delphi, this method contains a workaround for TGraphic bug.

CreateGraphicByType creates a graphic of the specified type, see "Choosing default graphic classes" above.

LoadFromFile creates a graphic object, loads the file FileName in it, and returns this graphic. A TGraphic class of a created object is determined by the file extension of the file. If the file extension is not recognized, or loading failed, the method returns *nil*.

LoadFromStream creates a graphic object, loads the stream content in it, and returns this graphic. A graphic format is recognized by content. If this procedure cannot recognize a graphic format, or no graphic class is defined for this graphic type,

See also

how to use third-party graphic classes in TRichView [671]

1.6.7 TRVIntegerList

List of integer values, used internally by TRichView classes.

Unit RVClasses;

```
type
  TRVIntegerList = class(TList)
```

Main properties:

Items[Index: Integer]: Integer.

Count: Integer

1.6.8 TRVRect

This is a class similar to TRect record.

Unit RVStyle;

Syntax

```
TRVRect = class (TPersistent)
```

(Introduced in version 1.2)

This class is used for representing offsets (padding) of colored background and border of paragraphs.

The class has Left, Top, Right and Bottom properties of TRVStyleLength 644 type.

1.6.9 TRVUnitsRect

This class contains 4 properties of TRVLength 636 type: Left, Top, Right, Bottom

Unit RVStyle;

Syntax

```
TRVFloatRect = class (TPersistent)
```

(Introduced in version 15)

This class is used for Margins 486 property of TRVPrint 479.

1.6.10 TRVThumbnailMaker

This class creates smooth scaled copies of images.

Unit RVThumbMaker;

Syntax

```
TRVThumbnailMaker = class;
```

(introduced in version 15)

Using

A single variable of this class is created: RVThumbnailMaker 664. You can change properties and call methods of the object returned in this variable.

Properties

MaxThumbnailSize: Integer. Maximal possible width and height of scaled images.

Methods

```
procedure AllowThumbnailsFor(Cls: TGraphicClass);
procedure DisallowThumbnailsFor(Cls: TGraphicClass);
function IsAllowedFor(Cls: TGraphicClass): Boolean;
function MakeThumbnail(Gr: TGraphic; Width, Height: Integer): TBitmap;
```

AllowThumbnailsFor allows making scaled images for the specified graphic class. By default, scaling is allowed for TBitmap and TJPEGImage. For Delphi 2010 or newer, scaling is allowed for TWicImage as well (used for TIFF images).

AllowThumbnailsFor disallows scaling for the specified graphic class.

IsAllowedFor tests if scaling allowed for the specified graphic class.

MakeThumbnail creates and returns a bitmap made from **Gr** and scaled to **Width** x **Height**. If Width or Height

exceeds **MaxThumbnailSize**, they are reduced proportionally. The method returns *nil* in the following cases:

- making scaled images is not allowed for the class of **Gr**;
- **Gr** is a transparent TBitmap;
- **Gr** is empty (0 x 0);
- **Gr**.Width=**Width** and **Gr**.Height=**Height**;
- both max(**Gr**.Width, **Gr**.Height) and max(**Width**, **Height**) exceed **MaxThumbnailSize**, and **Gr** is smaller than **Width** x **Height** (because we want to enlarge the image, but it would be shrunken instead)

See also

See also:

- smooth image scaling 57

1.7 Procedures and Functions

1.7.1 DrawControl and Others

These functions create bitmap having the size of the specified control, draw image of this control into this bitmap, return this bitmap.

Unit CtrlImg.

Syntax

```
function DrawControl(ctrl: TControl): TBitmap;
```

Also

```
function DrawButton(ctrl: TButton): TBitmap;
function DrawEdit(ctrl: TEdit): TBitmap;
function DrawMemo(ctrl: TMemo): TBitmap;
function DrawPanel(ctrl: TPanel): TBitmap;
```

DrawControl has the functionality of all other functions of this group.

TRVPrint use these functions for default printing of inserted controls 94, see OnPrintComponent 519 event.

1.7.2 GetRVESearchOptions

Converts options of TFindDialog/TReplaceDialog to TRichViewEdit 282 search options.

Unit RVMisc.

Syntax

```
function GetRVESearchOptions(fo: TFindOptions): TRVESearchOptions 626;
```

This function always includes *rvseoMultiItem* and *rvseoSmartStart* in the result.

See also

- TRichViewEdit.SearchText 336.

1.7.3 GetRVSearchOptions

Converts options of TFindDialog to TRichView 128 search options

Unit RVMisc.

Syntax

```
function GetRVSearchOptions(fo: TFindOptions): TRVSearchOptions 642;
```

This function always includes *rvsroMultiItem* and *rvsroSmartStart* in the result.

See also

- TRichView.SearchText 222.

1.7.4 RV_GetPrinterDC

Returns handle compatible with the the current printer object.

Unit PtblRV.

Syntax

```
function RV_GetPrinterDC: HDC;
```

After using this handle, delete it with DeleteDC.

Alternatively, you can use Printer.Handle.

Example

The procedure below returns size of document area on page, converted to screen pixels. Document area is paper size minus margins.

For conversion, it uses screen resolution (RichViewPixelsPerInch 655 is ignored).

It does not take headers 493 and footers 493 into account.

```
procedure GetPageSize(RVPrint: TRVPrint 479; var Width, Height: Integer);
var
  DC: HDC;
  phoX, phoY, phW, phH, lpy, lpx, LM, TM, RM, BM: Integer;
begin
  DC := RV_GetPrinterDC;

  Width := GetDeviceCaps(DC, HORZRES);
  Height := GetDeviceCaps(DC, VERTRES);

  lpy := GetDeviceCaps(DC, LOGPIXELSY);
  lpx := GetDeviceCaps(DC, LOGPIXELSX);

  phoX := GetDeviceCaps(DC, PHYSICALOFFSETX);
  phoY := GetDeviceCaps(DC, PHYSICALOFFSETY);
  phW := GetDeviceCaps(DC, PHYSICALWIDTH);
  phH := GetDeviceCaps(DC, PHYSICALHEIGHT);

  LM := RV_UnitsToPixels 622(RVPrint.Margins 486.Left,   FUnits, lpx)- phoX;
  TM := RV_UnitsToPixels(RVPrint.Margins.Top,    FUnits, lpy)- phoY;
  RM := RV_UnitsToPixels(RVPrint.Margins.Right,  FUnits, lpx)- (phW-(phoX+Width));
  BM := RV_UnitsToPixels(RVPrint.Margins.Bottom, FUnits, lpy)- (phH-(phoY+Height));

  if RVPrint.FixMarginsMode 484=rvfmmAutoCorrect then
  begin
    if LM<0 then
    begin
      inc(RM, LM);
      LM := 0;
    end;
    if TM<0 then
    begin
      inc(BM, TM);
      TM := 0;
    end;
```

```
    if RM<0 then
      RM := 0;
    if BM<0 then
      BM := 0;
  end;

  dec(Width, LM+RM);
  dec(Height, TM+BM);

  DeleteDC(DC);

  DC := GetDC(0);
  Width := MulDiv(Width, GetDeviceCaps(DC, LOGPIXELSX), lpx);
  Height := MulDiv(Height, GetDeviceCaps(DC, LOGPIXELSY), lpy);
  ReleaseDC(0, DC);
end;
```

Example of using (making document width in editor equal to page width, taking difference in resolutions into account):

```
var w, h: Integer;

// rvoClientTextWidth must be excluded from RichViewEdit1.Options.

GetPageSize(RVPrint1, w,h);
RichViewEdit1.MaxTextWidth := w-RichViewEdit1.LeftMargin-RichViewEdit1.RightMargin;
RichViewEdit1.MinTextWidth := RichViewEdit1.MaxTextWidth;
RichViewEdit1.Format;
```

1.7.5 RVGetFirstEndnote and Others

These functions allow to enumerate endnotes, footnotes, sidenotes and text box items in a document.

Unit RVNote.

```
function RVGetFirstEndnote(RichView: TCustomRichView): TRVEndnoteItemInfo;
function RVGetNextEndnote(RichView: TCustomRichView;
  Endnote: TRVEndnoteItemInfo): TRVEndnoteItemInfo;

function RVGetFirstFootnote(RichView: TCustomRichView): TRVFootnoteItemInfo;
function RVGetNextFootnote(RichView: TCustomRichView;
  Footnote: TRVFootnoteItemInfo): TRVFootnoteItemInfo;
```

(introduced in version 10)

Unit RVSidenote.

```
function RVGetFirstSidenote(RichView: TCustomRichView): TRVSidenoteItemInfo;
function RVGetNextSidenote(RichView: TCustomRichView;
  Sidenote: TRVSidenoteItemInfo): TRVSidenoteItemInfo;
```

RVGetFirstEndnote returns the first endnote in **RichView** (or *nil*, if **RichView** has no endnotes).

RVGetNextEndnote returns the next endnote after **Endnote** (or *nil*, if this is the last endnote).

RVGetFirstFootnote returns the first footnote in **RichView** (or *nil*, if **RichView** has no footnotes).

RVGetNextFootnote returns the next footnote after **Footnote** (or *nil*, if this is the last footnote).

RVGetFirstSidenote returns the first sidenote or text box item in **RichView** (or *nil*, if **RichView** has no sidenote and text box items).

RVGetNextFootnote returns the next sidenote or text box item after **Sidenote** (or *nil*, if this is the last sidenote

or text box item).

1.7.6 RVGetNoteTextStyleNo

Returns index of text style for endnote[103] or footnote[103].

Unit RVNote.

```
function RVGetNoteTextStyleNo(RVStyle: TRVStyle[385]; StyleNo: Integer;
  StyleTemplateName: TRVStyleTemplateName[644]=''): Integer;
```

(introduced in version 10, changed in version 14)

This function returns text style (index in **RVStyle**.TextStyles[404]) which can be used for footnote or endnote characters.

If **StyleTemplateName** is not empty, and a style template with this name[461] exists in **RVStyle**.StyleTemplates[402], the function returns the index of a text style having properties of the **StyleNo**-th text style after applying this style template. Otherwise, it returns the index of a text style having all properties of RVStyle.TextStyles[404] [StyleNo], but SubSuperScriptType[441] = *rvsssSuperScript*.

In all cases, Protection[439] of the returned text style is [*rvprDoNotAutoSwitch*].

If such text style does not exist, this function adds it to the end of **RVStyle**.TextStyles[404].

The recommended names of style templates to use with this function:

- 'endnote reference' for endnotes[103] and note references[108] inside endnote text[586];
- 'footnote reference' for footnotes[104] and note references[108] inside footnote text[586].

1.7.7 RVIsURL, RVIsEmail

These function return *True* if the string is URL or e-mail address.

Unit RVFuncs.

```
function RVIsURL(const s: String): Boolean;
function RVIsEmail(const s: String): Boolean;
```

(introduced in version 10)

First, RVIsURL calls RVIsCustomURL[662]. If it returns *False*, RVIsURL checks for the following prefixes:

- http://
- ftp://
- file://
- gopher://
- mailto:
- https://
- news:
- telnet:
- wais:
- 'www.
- ftp.

In TRichView, this function is used to ignore URLs in live spelling check[69].Besides, it is used in demo projects and in ***RichViewActions***.

See also: how to create a chat window[673].

1.7.8 Unicode Conversion

Various Unicode[64]-related functions.

Unit RVUni.

```
function RVU_RawUnicodeToWideString(const s: TRVRawByteString[638]):TRVUnicodeString[647];
function RVU_GetRawUnicode(const s: TRVUnicodeString[647]):TRVRawByteString[638];
```

RVU_RawUnicodeToWideString converts "raw Unicode" string to TRVUnicodeString.

RVU_GetRawUnicode converts TRVUnicodeString to "raw Unicode" string.

```
type
  TRVCodePage = type cardinal; // defined in RVStyle unit
function RVU_AnsiToUnicode(CodePage: TRVCodePage; const s: TRVAnsiString[623]): TRVRawByteStri
function RVU_UnicodeToAnsi(CodePage: TRVCodePage; const s: TRVRawByteString[638]): TRVAnsiStri
function RVU_AnsiToUTF8(CodePage: TRVCodePage; const s: TRVAnsiString[623]): TRVRawByteString
```

RVU_AnsiToUnicode converts ANSI string (in the specified CodePage) to "raw Unicode" string.

RVU_UnicodeToAnsi converts "raw Unicode" string to ANSI string (in the specified CodePage).

RVU_AnsiToUTF8 converts ANSI string (in the specified CodePage) to UTF-8 string.

```
type
  TRVUnicodeTestResult = (rvutNo, rvutYes, rvutProbably, rvutEmpty, rvutError);
function RV_TestStreamUnicode(Stream: TStream): TRVUnicodeTestResult;
function RV_TestFileUnicode(const FileName: String): TRVUnicodeTestResult;
function RV_TestStringUnicode(const s: TRVRawByteString[638]): TRVUnicodeTestResult;
```

These functions do very basic test of stream, file or string content to determine if they contain Unicode or ANSI text.

Possible results:

- *rvutNo* – it is ANSI text;
- *rvutYes* – it is (very very probably) Unicode text;
- *rvutProbably* – it may be either ANSI or Unicode;
- *rvutEmpty* – empty content;
- *rvutError* – error (reading file or stream).

First, these function check number of bytes in content. If it is odd, this is not Unicode text (*rvutNo*).

Next, they check the first 2 bytes of text. If it is a Unicode byte order mark, this is very probably Unicode (*rvutYes*).

Next, they check if there are 0-bytes in the first 500 bytes. If yes, it is not an ANSI (*rvutYes*), otherwise it may be any (*rvutProbably*).

You can also use WinAPI function **IsTextUnicode** performing more advanced tests.

See also:

- Unicode in TRichView[64].

1.7.9 Functions from RVGetText Unit

The units RVGetText and RVGetTextW include the identical sets of functions returning text from TRichView.

Functions from RVGetText return ANSI strings, functions from RVGetTextW return Unicode strings.

RVGetText:

```
function GetCurrentLineText(rve: TCustomRichViewEdit[282]): TRVAnsiString[623];
function GetVisibleText(rv: TCustomRichView[128]): TRVAnsiString[623];
function GetAllText(rv: TCustomRichView[128]): TRVAnsiString[623];
function GetRVDataText(RVData: TCustomRVData[608]): TRVAnsiString[623];
function GetCurrentParaSectionText(rve: TCustomRichViewEdit[282]): TRVAnsiString[623];
```

```
function GetCurrentParaText(rve: TCustomRichViewEdit[282]): TRVAnsiString[623];
function GetCurrentChar(rve: TCustomRichViewEdit[282]): TRVAnsiString[623];
function GetCurrentWord(rve: TCustomRichViewEdit[282]): TRVAnsiString[623];
```

RVGetTextW:

```
function GetCurrentLineText(rve: TCustomRichViewEdit[282]): TRVUnicodeString[647];
function GetVisibleText(rv: TCustomRichView[128]): TRVUnicodeString[647];
function GetAllText(rv: TCustomRichView[128]): TRVUnicodeString[647];
function GetRVDataText(RVData: TCustomRVData[608]): TRVUnicodeString[647];
function GetCurrentParaSectionText(rve: TCustomRichViewEdit[282]): TRVUnicodeString[647];
function GetCurrentParaText(rve: TCustomRichViewEdit[282]): TRVUnicodeString[647];
function GetCurrentChar(rve: TCustomRichViewEdit[282]): TRVUnicodeString[647];
function GetCurrentWord(rve: TCustomRichViewEdit[282]): TRVUnicodeString[647];
```

GetCurrentLineText returns the current line of text (the line containing the caret).

GetAllText returns all text in rv, **GetRVDataText** returns all text in RVData.

GetCurrentParaSectionText returns text of the paragraph section with the caret (paragraph section is a part of document limited either by paragraph breaks or by new line breaks (added with Shift+Enter).

GetCurrentParaText returns text of the paragraph with the caret.

GetCurrentChar returns character to the left of the caret. If the caret is at the beginning of line, returns the character to the right. If no character, returns empty string.

GetCurrentWord returns the word with the caret.

For non-text items, these function return their text representation.

See also:

- RVGetTextRange[621]

1.7.10 Functions from RVLinear Unit

```
const RVCharsPerLineBreak: Integer = 1;
      RVNonTextCharacter: Char = ' ';
```

```
function RVGetLinearCaretPos(rve: TCustomRichViewEdit[282]): Integer;
procedure RVSetLinearCaretPos(rve: TCustomRichViewEdit[282]; LinearPos: Integer);
```

RVGetLinearCaretPos returns the caret position (as a number of characters before the caret; non-text objects (except for the tables[98] and tabulators[89]) are treated as one character, line breaks are treated as RVCharsPerLineBreak characters). **RVSetLinearCaretPos** moves the caret to the specified position.

```
procedure RVGetSelection(rv: TCustomRichView[128]; var SelStart, SelLength: Integer);
procedure RVSetSelection(rv: TCustomRichView[128]; SelStart, SelLength: Integer);
```

RVGetSelection returns the selection position in richedit-like parameters (SelStart and SelLength). **RVSetSelection** sets selection by these values.

```
type
  TRVSelection = record
    SelStart, SelLength: Integer;
    MultiCell: Boolean;
    StartRow, StartCol, RowOffs, ColOffs: Integer;
  end;
procedure RVGetSelectionEx(rv: TCustomRichView[128]; var Selection: TRVSelection);
procedure RVSetSelectionEx(rv: TCustomRichView[128]; const Selection: TRVSelection);
```

RVGetSelectionEx returns the selection position in TRVSelection record. **RVSetSelectionEx** sets selection by this record. These function can handle not only a normal selection, but a multicell selection in tables as well.

```
function RichViewToLinear(rv: TCustomRichView[128]; CurRVData, RVData: TCustomRVData[608];
  ItemNo, ItemOffs: Integer; var LinearPos: Integer): Boolean;
function LinearToRichView(rv: TCustomRichView[128]; CurRVData: TCustomRVData[608];
  var LinearPos: Integer; var RVData: TCustomRVData[608];
  var ItemNo, ItemOffs: Integer): Boolean;
```

RichViewToLinear converts the position specified in RichView coordinates (RVData – document, ItemNo – index of item in this document, ItemOffs – offset in this item) to linear coordinate (a number of characters from the beginning of CurRVData before this position; non-text objects (except for the tables[98] and tabulators[89]) are treated as one character, line breaks are treated as RVCharsPerLineBreak characters). **LinearToRichView** performs an opposite conversion.

```
function RVGetTextRange(rv: TCustomRichView[128]; RangeStart, RangeLength: Integer): String;
function RVGetTextLength(rv: TCustomRichView[128]): Integer;
```

RVGetTextRange returns text from rv, from RangeStart to RangeLength position. Non-text items (except for the tables[98] and tabulators[89]) are returned as RVNonTextCharacter. If RVCharsPerLineBreak=1, #13 is used as line break. If RVCharsPerLineBreak=2, #13#10 is used as line break.

The call RVGetTextRange(rv, 0, RVGetTextLength(rv)) returns the whole text.

All these functions are compatible, they calculate position identically. The text returned by RVGetTextRange has one-to-one correspondence to RichView document (unlike text returned by the functions from RVGetText[620] unit).

Unicode note: for Delphi/C++Builder 2009 or newer, RVGetTextRange returns Unicode text. For older versions of Delphi, it returns ANSI text.

1.7.11 Functions for TRVUnits conversion

unit RVFuncs;

```
function RV_UnitsPerInch(Units: TRVUnits[648]): TRVLength[636];
function RV_UnitsToTwips(Value: TRVLength[636]; Units: TRVUnits[648]): Integer;
function RV_TwipsToUnits(Value: Integer; Units: TRVUnits[648]): TRVLength[636];
function RV_UnitsToUnits(Value: TRVLength; OldUnits, NewUnits: TRVUnits[648]): TRVLength[636];
function RV_UnitsToPixels(Value: TRVLength[636]; Units: TRVUnits[648]; PixelsPerInch: Integer):
```

RV_UnitsPerInch returns the amount of Units in 1 inch.

RV_UnitsToTwips converts Value from Units to twips.

RV_TwipsToUnits converts Value from twips to Units.

RV_UnitsToUnits converts Value from OldUnits to NewUnits.

RV_UnitsToPixels converts Value from Units to pixels in the specified resolution.

See also

- Units of measurement in TRichView[74]

1.8 Types

1.8.1 TCheckpointData

Unit RVStyle;

```
type
  TCheckpointData = type Pointer;
```

Value of this type identifies one *checkpoint*[30].

It can be obtained as a parameter of several events, or as a return value of several methods of TRichView.

All properties of checkpoint can be extracted from this value using other RichView methods.

See "Checkpoints" 30 for details.

1.8.2 TRVAnsiString

Unit RVTypes.

Declaration for Delphi 2009 or newer:

```
type
  TRVAnsiString = type AnsiString;
```

Declaration for older versions of Delphi:

```
type
  TRVAnsiString = type String;
```

Strings of this type contain ANSI text.

See also

- TRVUnicodeString 647
- TRVRawByteString 638

1.8.3 TRVBiDiMode

Unit RVStyle;

```
type
  TRVBiDiMode =
    (rvbdUnspecified, rvbdLeftToRight, rvbdRightToLeft);
```

Value of this type defines default text reading direction for document (TRichView.BiDiMode 137), paragraph (TParaInfo.BiDiMode 450), or text (TFontInfo.BiDiMode 433).

Value	Meaning
rvbdUnspecified	Bidirectional text is not supported (if specified for document), or text direction is inherited from higher level: ▪ for text – from paragraph; ▪ for paragraph – from document.
rvbdLeftToRight	Default text reading order is left-to-right.
rvbdRightToLeft	Default text reading order is right-to-left.

See also:

Bidirectional text in TRichView 69

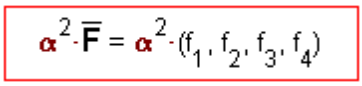
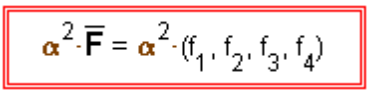
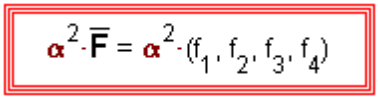
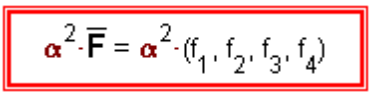
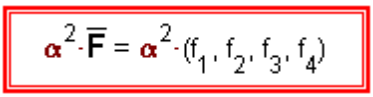
1.8.4 TRVBorderStyle

Unit RVStyle

```
type
  TRVBorderStyle = (rvbNone, rvbSingle, rvbDouble, rvbTriple,
    rvbThickInside, rvbThickOutside);
```

Style of border around paragraph 39 or around a text box (of a sidenote 105 or a text box item 107). Type for Border 451.Style 468 property.

Style	Border

rvbNone	No border
rvbSingle	
rvbDouble	
rvbTriple	
rvbThickInside	
rvbThickOutside	

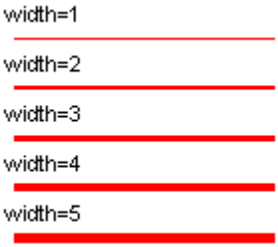
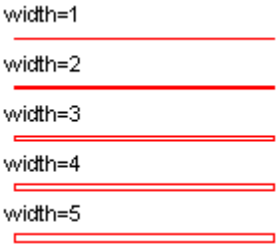
(examples are made with Color[468]=*clRed*, Width[468]=1, InternalWidth[468]=1)

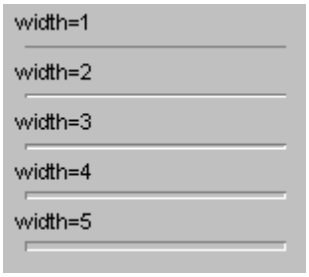
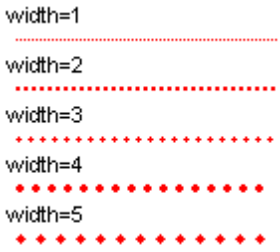
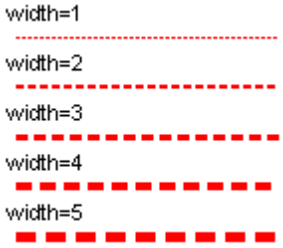
1.8.5 TRVBreakStyle

Unit RVStyle

```
type
  TRVBreakStyle =
    (rvbsLine, rvbsRectangle,
     rvbs3d, rvbsDotted, rvbsDashed);
```

Style of *break*[93] item.

Style	Example
rvbsLine	
rvbsRectangle	

rvbs3d	 (break color is ignored)
rvbsDotted	
rvbsDashed	

1.8.6 TRVCellVAlign

Unit RVTable;

```
type
  TRVCellVAlign =
    (rvcTop,  rvcMiddle, rvcBottom, rvcVDefault);
```

Types vertical alignment in cells.

See VAlign[578] property of table rows and VAlign[575] property of cells.

1.8.7 TRVCodePage

Unit RVStyle;

```
type
  TRVCodePage = type Cardinal;
```

A type for code page identifiers.

1.8.8 TRVColorMode

Unit RVStyle;

```
type
  TRVColorMode = (rvcmColor, rvcmPrinterColor,
    rvcmGrayScale, rvcmBlackAndWhite, rvcmBlackOnWhite);
```

Mode	Meaning
rvcmColor	Color output.
rvcmPrinterColor	Color output, but the following colors are changed: ▪ *clWindow*, *clBtnHighlight* to *clWhite*, ▪ *clWindowText* to *clBlack*, ▪ *clBtnShadow* to *clGray*, ▪ *clBtnFace*, *clScrollbar* to *clSilver*.
rvcmGrayScale	Colors are converted to shades of gray.
rvcmBlackAndWhite	Paragraphs backgrounds are not printed; text is printed black on white or white on black depending on what is darker.
rvcmBlackOnWhite	Paragraphs backgrounds are not printed; text is printed black on white.

In the current version of the components, color mode affects drawing and printing text, paragraph background, *breaks* 93, list markers 99, tables 98.

It does not affect drawing and printing pictures.

1.8.9 TRVDisplayOptions

Unit RVScroll

```
type
  TRVDisplayOption = (rvdoImages, rvdoComponents, rvdoBullets);
  TRVDisplayOptions = set of TRVDisplayOption;
const
  rvdoALL = [rvdoImages, rvdoComponents, rvdoBullets];
```

1.8.10 TRVDocRotation

Unit RVStyle;

```
type
  TRVDocRotation = (rvrotNone, rvrot90, rvrot180, rvrot270);
```

Possible values for a table cell rotation.

Value	Meaning
rvrotNone	Not rotated (0°)
rvrot90	Rotated by 90°
rvrot180	Rotated by 180°
rvrot270	Rotated by 270°

1.8.11 TRVESearchOptions

Unit RVEdit;

```
type
  TRVESearchOption = (rvseoMatchCase, rvseoDown, rvseoWholeWord,
```

```
    rvseoMultiItem, rvseoSmartStart);
  TRVESearchOptions = set of TRVESearchOption;
```

Options for searching in RichViewEdit, see TCustomRichViewEdit.SearchText[336].

Value	Meaning
rvseoMatchCase	If included, character case is taken into account when comparing string.
rvseoDown	If included, search is performed to the end of document. If not included, it is performed to the top of document.
rvseoWholeWord	If included, the searched string matches only whole words.
rvseoMultiItem	If included, the search can match substrings of several text items[88]. If not included, the text is searched in each text item separately. For example, if the document contains the text **He*llo***, the substring 'Hello' can be found only if this option is included, because 'He' and 'llo' belong to different items.
rvseoSmartStart	If included, and a document fragment is selected, then: ▪ when searching down, the search starts from the second selected character; ▪ when searching up, the search starts from the last but one selected character (selected characters are counted from the top of the document, the selection direction is ignored)

1.8.12 TRVExtraItemProperty

Unit RVItem;

```
type
 TRVExtraItemProperty =
  (rvepUnknown, rvepVShift, rvepVShiftAbs,
   rvepImageWidth, rvepImageHeight,
   rvepTransparent, rvepTransparentMode,
   rvepTransparentColor, rvepMinHeightOnPage,
   rvepSpacing, rvepResizable,
   rvepDeleteProtect, rvepNoHTMLImageSize,
   rvepAnimationInterval, rvepVisible,
   rvepHidden, rvepShared,
   rvepBorderWidth, rvepColor, rvepBorderColor,
   rvepOuterHSpacing, rvepOuterVSpacing,
   rvepVAlign);
```

Identifies an item property of integer type.

Positioning

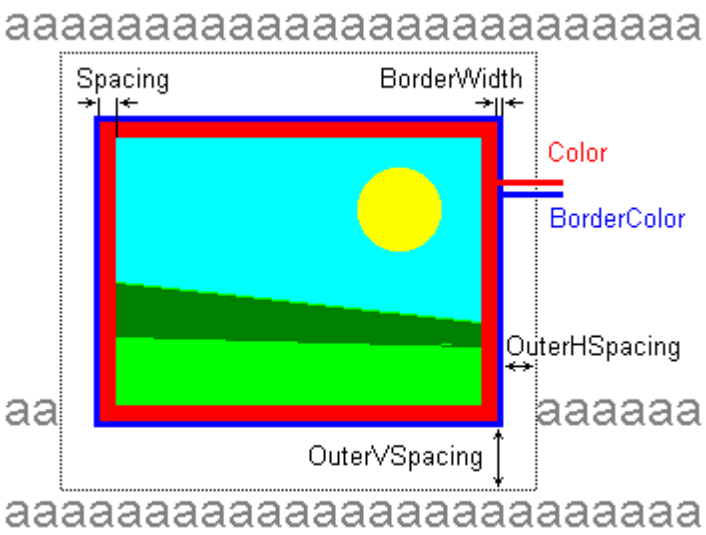
Property	Applicable To	Default Value	Meaning
rvepVShift	rect.items*	0	Vertical offset, in % of item height (default), or in RVStyle.Units[404] (see *rvepVShiftAbs*).

			Negative values move the item down, positive move it up (relative to the position specified in VAlign[648])
rvepVShiftAbs	rect.items*	0	If non-zero, *rvepVShift* defines vertical offset in RVStyle.Units[404] instead of %.
rvepVAlign	rect.items*	ord(*rvvaBaseline*)	Vertical alignment, TRVVAlign[648] converted to integer.

* "rect.items" include pictures[90], *hot-pictures*[93], controls[94], *bullets*[96], *hotspots*[97], labels[101], sequences[102], footnotes[104], endnotes[103], note references[108].

Borders and spacing

Scheme:



Property	Applicable To	Default Value	Meaning
rvepSpacing	rect.items*	1	Spacing around the item content, in RVStyle.Units[404]. Must be a zero or a positive value. This is a padding, a spacing between the item content and the item border. This area may be colored with *rvepColor*.
rvepColor	rect.items*	*clNone*	Background color. It is drawn below the image (and around the image, if *rvepSpacing* is positive).
rvepBorderWidth	rect.items*	0	Border width, in RVStyle.Units[404]. Must be a zero or a positive value.
rvepBorderColor	rect.items*	*clBlack*	Border color.
rvepOuterHSpacing *rvepOuterVSpacing*	rect.items*	0	Spacing around the border (horizontal and vertical), in RVStyle.Units[404]. Must be a zero or a positive value.

* "rect.items" include pictures[90], *hot-pictures*[93], controls[94], *bullets*[96], *hotspots*[97], labels[101], sequences[102], footnotes[104], endnotes[103], note references[108].

Pictures

Property	Applicable To	Default Value	Meaning
rvepImageWidth, *rvepImageHeight*	▪ pictures 90 ▪ *hot-pictures* 93	0	If set to positive value, define size of the image in RVStyle.Units 404. The actual image is not modified, it is just drawn stretched. See also *rvepAnimationInterval* property below.
rvepTransparent, *rvepTransparentMode*, *rvepTransparentColor*	▪ pictures 90 ▪ *hot-pictures* 93 (if they are TBitmap)		Correspond to properties of TBitmap: Transparent, TransparentMode, TransparentColor. Limitations: ▪ transparency is turned off in 256-color mode; ▪ transparency is not supported when printing (although print preview shows transparent bitmaps) ▪ TBitmap implementation of transparency is not ideal; sometimes it may cause weird colors for 256-color images (probably, it was fixed in last versions of Delphi)
rvepMinHeightOnPage	▪ pictures 90 ▪ *hot-pictures* 93	0	Affects printing, allows printing large pictures on several pages. Positive value defines minimal height of part of image allowed to be on page. For example: ▪ 1 – the image can be printed on two or more pages, it can separated between pages in any place; ▪ 100 – if the rest of page can contain >= 100 pixels of the image, the image will be separated, otherwise it will be printed on single page; ▪ 0 – the image/control will be printed on single page. Do not overuse this feature, it has limitations: ▪ such images/controls are always printed from new line; ▪ for control, OnPrintComponent 519 must return bitmap having exactly the size of the control (if this property = 0, you can return larger bitmap and it will be stretched allowing to create a better printing output)
rvepNoHTMLImageSize	▪ pictures 90 ▪ *hot-pictures* 93 ▪ *bullets* 96 ▪ *hotspots* 97 ▪ list markers 99	0	If non-zero, image size is not saved in HTML, even if *rvsoImageSizes* is included in the Options parameter for SaveHTML 216/SaveHTMLEx 217 This option is ignored for pictures 90 and *hot-pictures* 93, if *rvepImageWidth* or *rvepImageHeight* properties are non-zero.
rvepAnimationInterval	▪ pictures 90 ▪ *hot-pictures* 93 (if they are TBitmap)	0	Animation delay in 1/100 of second. If nonzero and *rvepImageWidth* and/or *rvepImageHeight* are defined, playing bitmap animation 56 (in imagewidth x imageheight frame)

Editing

Property	Applicable To	Default Value	Meaning
rvepResizable	▪ pictures 90 ▪ *hot-pictures* 93 ▪ controls 94	1 (for pictures and *hot-pictures*), 0 (for controls).	Allows resizing with mouse ▪ 0 – no resizing; ▪ non-zero value – resizing is allowed.
rvepDeleteProtect	all non-text items	0	▪ 0 – no protection; ▪ non-zero value – item cannot be deleted in editing operations. See also: *rvprDeleteProtect* option of Protection 439 for text style.

Other

Property	Applicable To	Default Value	Meaning
rvepVisible	▪ controls 94	1	Set to 0 to hide the control (it still occupies its place, but is not displayed)
rvepHidden	all non-text items	0	If non-zero, this item is hidden 42.
rvepShared	▪ pictures 90 ▪ *hot-pictures* 93 ▪ controls 94	0	If non-zero, the item content (picture or control) is not deleted when the item is deleted.

See also

- TRVExtraItemStrProperty 630

See also methods of TRichView

- SetItemExtraIntProperty 230,
- GetItemExtraIntProperty 193.

See also methods of TRichViewEdit

- SetItemExtraIntPropertyEd 350,
- SetCurrentItemExtraIntProperty 345,
- GetCurrentItemExtraIntProperty 312.

1.8.13 TRVExtraItemStrProperty

Unit RVItem;

```
type
  TRVExtraItemStrProperty = (
    rvespUnknown,
    rvespHint,
    rvespAlt,
    rvespImageFileName,
    rvespTag);
```

Identifies item property of string type.

Default value for all properties is '' (empty string).

Property	Applicable To	Meaning

rvespUnknown		Not a property. For internal use.
rvespAlt	▪ pictures 90 ▪ *hot-pictures* 93 ▪ *bullets* 96 ▪ *hotspots* 97	Text representation of image. Saved as "<img alt>" attribute in HTML files. Must not contain line breaks and double quotes.
rvespHint	all items (but not shown for controls and tables)	Popup hint can be displayed for items if *rvoShowItemHints* is included in RichView.Options 146 and RichView.ShowHint=*True*. Hints are displayed by assigning to RichView.Hint. See also OnItemHint 246 event. Note: this feature adds 4 bytes in memory to each item. If you do not need it, enable RVDONOTUSEITEMHINTS compiler define in RV_Defs.inc. After enabling this define, you can still use OnItemHint 246. Hint string must not contain line breaks and double quotes. When exporting to HTML, it is exported as "title" attribute. For RTF, hints are exported and imported only for hyperlinks. In OnWriteHyperlink 267 event, the proper value of Extras parameter is set by default.
rvespImageFileName	▪ pictures 90 ▪ *hot-pictures* 93 ▪ tables 98	Provides a place for storing image file name. For tables, it is a background image, also accessible as table.BackgroundImageFileName 531. See also: TRVTableCellData.BackgroundImageFileName 568 If *rvsoUseItemImageFileNames* is included in the Options parameter of SaveHTML 216/SaveHTMLEx 217, images with defined (non-empty) file names will not be saved, but their file names will be written in HTML (paths will be converted to relative paths).
rvespTag	all items	tag 34

See also

- TRVExtraItemProperty 627

See also methods of TRichView

- SetItemExtraStrProperty 231,
- GetItemExtraStrProperty 193.

See also methods of TRichViewEdit

- SetItemExtraStrPropertyEd 351,
- SetCurrentItemExtraStrProperty 346,
- GetCurrentItemExtraStrProperty 313.

1.8.14 TRVFloatPosition

Unit RVFloatingPos;

```
type
  TRVFloatPosition = type Integer;
```

This is a type of HorizontalOffset[597] and VerticalOffset[600] properties of TRVBoxPosition[596].

Meaning of these values depend on HorizontalPositionKind[597] and VerticalPositionKind[600] properties, and may mean either values in TRVStyleUnits[644] or in (1/1000)%.

1.8.15 TRVFloatPositionKind

Unit RVFloatingPos;

Values of this type define how a position of a floating box is calculated.

```
type
  TRVFloatPositionKind = (rvfpkAlignment, rvfpkAbsPosition, rvfpkPercentPosition);
```

This is a type of the following properties of TRVBoxPosition[596]:

- HorizontalPositionKind[597]
- VerticalPositionKind[600]

Value	Meaning
rvfpkAlignment	A position is specified in HorizontalAlignment[597]/VerticalAlignment[600]
rvfpkAbsPosition	HorizontalOffset[597]/VerticalOffset[600] contains offset in RVStyle.Units[404] (pixels or twips)
rvfpkPercentPosition	HorizontalOffset[597]/VerticalOffset[600] contains offset in (1/1000)% of the anchor area size

1.8.16 TRVFloatProperty

Unit RichView.

Identifies a property of TRichView[128] of a TRVLength[636] type.

```
type
  TRVFloatProperty = (rvfpDPLeftMargin, rvfpDPRightMargin, rvfpDPTopMargin,
    rvfpDPBottomMargin, rvfpDPHeaderY, rvfpDPFooterY);
```

Value	Corresponding property of TRichView
rvfpDPLeftMargin	DocParameters[140].LeftMargin[278]
rvfpDPRightMargin	DocParameters.RightMargin[279]
rvfpDPTopMargin	DocParameters.TopMargin[280]
rvfpDPBottomMargin	DocParameters.BottomMargin[277]
rvfpDPHeaderY	DocParameters.HeaderY[278]
rvfpDPFooterY	DocParameters.FooterY[278]

A parameter of this type is used in TRichViewEdit[282].SetFloatPropertyEd[337].

See also:

- TRVIntProperty 635;
- TRVStrProperty 643.

1.8.17 TRVFloatSize

Unit RVFloatingBox;

```
type
  TRVFloatSize = type Integer;
```

This is a type of Width 595 and Height 594 properties of TRVBoxProperties 593.

Meaning of these values depend on WidthType 595 and HeightType 594 properties, and may mean either values in TRVStyleUnits 644 or in (1/1000)%.

1.8.18 TRVFontInfoProperties

Unit RVStyle;

```
type
  TRVFontInfoProperty = (
    rvfiFontName, rvfiSize, rvfiCharset, rvfiUnicode,
    rvfiBold, rvfiItalic, rvfiUnderline, rvfiStrikeout,
    rvfiOverline, rvfiAllCaps, rvfiSubSuperScriptType,
    rvfiVShift, rvfiColor, rvfiBackColor,
    rvfiJump, rvfiHoverBackColor, rvfiHoverColor, rvfiHoverUnderline,
    rvfiJumpCursor,
    rvfiNextStyleNo, rvfiProtection, rvfiCharScale, rvfiBaseStyleNo,
    rvfiBiDiMode, rvfiCharSpacing, rvfiHTMLCode, rvfiRTFCode, rvfiHidden,
    {$IFDEF RVLANGUAGEPROPERTY}
    rvfiLanguage,
    {$ENDIF}
    rvfiUnderlineType, rvfiUnderlineColor, rvfiHoverUnderlineColor,
    rvfiCustom, rvfiCustom2, rvfiCustom3, rvfiCustom4, rvfiCustom5);

TRVFontInfoProperties = set of TRVFontInfoProperty;
```

Identifies properties of TFontInfo 444.

This is a type of the following properties:

- TRVStyleTemplate.ValidTextProperties 464;
- TFontInfo.ModifiedProperties 445.

Parameters of this type are used in the following methods:

- TFontInfos.FindSuchStyle, FindSuchStyleEx 422;
- TFontInfo.IsEqual 448.

```
const
  RVAllFontInfoProperties: TRVFontInfoProperties =
    [Low(TRVFontInfoProperty)..High(TRVFontInfoProperty)];
```

Constant, lists all TFontInfo properties.

1.8.19 TRVFReaderStyleMode

Unit RVStyle;

```
type
  TRVFReaderStyleMode =
  (rvf_sIgnore, rvf_sInsertMap, rvf_sInsertMerge)
```

Defines a mode for loading collection of styles from RVF [59] (if they are contained there, see *rvfoSaveTextStyles* and *rvfoSaveParaStyles* RVFOptions [152]).

This is a type of the following properties of TCustomRichView:

- RVFTextStylesReadMode [154] (loading Style [156].TextStyles [404]);
- RVFParaStylesReadMode [154] (loading Style.ParaStyles [398] and Style.ListStyles [396]).

Mode	When loading RVF...	When Inserting RVF...
rvf_sIgnore	Collections of styles in RVF are ignored. For correct loading, this RVF file must be saved with the same collections of styles.	
rvf_sInsertMerge	Current collection of styles will be replaced with values from RVF.	Current styles will be merged with styles from RVF (new styles will be added to the collection, if necessary).
rvf_sInsertMap	Current collection of styles will be replaced with values from RVF.	Styles from RVF will be mapped to the most similar existing styles. New styles will not be added.

You should be careful with the last two modes: it's recommended to use them only if TRVStyle [385] component is used exclusively by the given TRichView[Edit].

If style templates are used [158], and RVFTextStylesReadMode [154]=RVFParaStylesReadMode [154] =*rvf_sInsertMerge*, and StyleTemplateInsertMode [156]<>*rvstimIgnoreSourceStyleTemplates*, the RVF loading methods replace the existing Style [156].StyleTemplates [402] with the style templates read from RVF; the RVF insertion methods merge the collection of style templates from RVF into the existing style templates, and read text and paragraph styles according to StyleTemplateInsertMode [156].

Loading operations include:

- LoadRVF [211],
- LoadRVFFromStream [210],
- loading from database fields [373].

Insertion operations include:

- InsertRVFFromStream [206],
- AppendRVFFromStream [173],
- TRichViewEdit.InsertRVFFromFileEd [324],
- TRichViewEdit.InsertRVFFromStreamEd [325],
- pasting [48] from the Clipboard,
- drag&drop [55].

1.8.20 TRVHFType

Type of header or footer

```
type
  TRVHFType = (rvhftNormal, rvhftFirstPage, rvhftEvenPages);
```

Value	Meaning
rvhftNormal	Header/footer for all pages (except for pages having a different header/footer specified)
rvhftFirstPage	Header/footer for the first page
rvhftEvenPages	Header/footer for even pages

1.8.21 TRVHTMLLength

Unit RVTable;

```
type
  TRVHTMLLength = type Integer;
```

Represents length either in percent or in TRVStyleUnits 644.

Range	Meaning
>0	Length in TRVStyleUnits 644 (twips or screen pixels)
<0	Length in percent (for example, -50 means 50%)
=0	Length is undefined (default)

See also

- TRVTableItemInfo.BestWidth 531
- TRVTableCellData.BestWidth 569

1.8.22 TRVIntProperty

Unit RichView.

Identifies a property of TRichView 128 of an integer or an ordinal type.

```
type
  TRVIntProperty = (rvipLeftMargin, rvipRightMargin, rvipTopMargin,
    rvipBottomMargin, rvipMaxTextWidth, rvipMinTextWidth, rvipBackgroundStyle,
    rvipColor, rvipBiDiMode, rvipDPUnits, rvipDPOrientation,
    rvipDPTitlePage, rvipDPFacingPages, rvipDPMirrorMargins);
```

Value	Corresponding property of TRichView
rvipLeftMargin	LeftMargin 145
rvipRightMargin	RightMargin 149
rvipTopMargin	TopMargin 158
rvipBottomMargin	BottomMargin 137
rvipMaxTextWidth	MaxTextWidth 146
rvipMinTextWidth	MinTextWidth 146
rvipBackgroundStyle	BackgroundStyle 136
rvipColor	Color 139
rvipBiDiMode	BiDiMode 137
rvipDPUnits	DocParameters 140.Units 280
rvipDPOrientation	DocParameters.Orientation 279
rvipDPTitlePage	DocParameters.TitlePage 280
rvipDPFacingPages	DocParameters.FacingPages 278
rvipDPMirrorMargins	DocParameters.MirrorMargin 279

A parameter of this type is used in TRichViewEdit[282].SetIntPropertyEd[337].

See also:

- TRVFloatProperty[632];
- TRVStrProperty[643].

1.8.23 TRVItemBackgroundStyle

Unit RVStyle;

```
type
  TRVItemBackgroundStyle =
    (rvbsColor, rvbsStretched, rvbsTiled, rvbsCentered);
```

Defines background style (position of background image).

Value	Meaning
rvbsColor	No image.
rvbsStretched	Image stretched to fill the whole area. Not all image types can be stretched.
rvbsTiled	Image is repeated to fill the whole area.
rvbsCentered	Image is located at the center.

See also:

- TRVTableItemInfo.BackgroundStyle[531] (background of table)
- TRVTableCellData.BackgroundStyle[568] (background of cell)
- Smooth image scaling[57]

1.8.24 TRVLength

Unit RVStyle;

```
type
  TRVLength = type Extended;
```

This is a type of values measured in TRVUnits[648].

1.8.25 TRVOleDropEffects

Type of drag&drop operation.

```
type
  TRVOleDropEffect = (rvdeNone, rvdeCopy, rvdeMove, rvdeLink);
  TRVOleDropEffects = set of TRVOleDropEffect;
```

This type defines how drag&drop cursor looks like.

Effect	Meaning
rvdeNone	Drag&drop is not possible
rvdeCopy	The source object will be copied to the target
rvdeMove	The source object will be moved to the target
rvdeLink	Link to the source object will be inserted in the target

See events of TCustomRichViewEdit:

- OnOleDragEnter[361];
- OnOleDragOver[362];
- OnOleDrop[363].

1.8.26 TRVOpacity

Unit RVStyle;

```
type
  TRVOpacity = type Integer;
```

Specifies the object opacity in 1/1000 of percent. Valid values are in range 0..100 000. The larger the value, the less transparent the object is.

For example:

- 0 means full transparency, the object is invisible
- 100 000 means that the object is opaque
- 50 500 means 50.5% opacity.

Physically, only 256 gradations of transparency is supported by screen and other devices. The higher precision range is chosen for the convenience of specifying opacity in fractions of percent.

1.8.27 TRVPageNumberType

Unit RVFieldItems.

```
type
  TRVPageNumberType = (rvpntDefault, rvpntDecimal, rvpntDecimalDash,
    rvpntLowerAlpha, rvpntUpperAlpha,
    rvpntLowerRoman, rvpntUpperRoman);
```

Numbering type for "page number" field[109].

Type	Meaning
rvpntDefault	Default numbering. Displayed as decimal.
rvpntDecimal	1, 2, 3,...
rvpntDecimalDash	- 1 -, - 2 -, - 3 -
rvpntLowerAlpha	a, b, c, ..., z, aa, ab, ...
rvpntUpperAlpha	A, B, C, ..., Z, AA, AB, ...
rvpntLowerRoman	i, ii, iii, iv, ...
rvpntUpperRoman	I, II, III, IV, ...

This is a type of the following properties:

- TRVPageNumberType.NumberType[605].

1.8.28 TRVParaInfoProperties

Unit RVStyle;

```
type
  TRVParaInfoProperty = (
    rvpiFirstIndent, rvpiLeftIndent, rvpiRightIndent,
    rvpiSpaceBefore, rvpiSpaceAfter, rvpiAlignment, rvpiLastLineAlignment,
```

```
    rvpiNextParaNo, rvpiDefStyleNo, rvpiLineSpacing, rvpiLineSpacingType,
    rvpiBackground_Color,
    rvpiBackground_BO_Left, rvpiBackground_BO_Top,
    rvpiBackground_BO_Right, rvpiBackground_BO_Bottom,
    rvpiBorder_Color, rvpiBorder_Style,
    rvpiBorder_Width, rvpiBorder_InternalWidth,
    rvpiBorder_BO_Left, rvpiBorder_BO_Top,
    rvpiBorder_BO_Right, rvpiBorder_BO_Bottom,
    rvpiBorder_Vis_Left, rvpiBorder_Vis_Top,
    rvpiBorder_Vis_Right, rvpiBorder_Vis_Bottom,
    rvpiNoWrap, rvpiReadOnly, rvpiStyleProtect, rvpiDoNotWantReturns,
    rvpiKeepLinesTogether, rvpiKeepWithNext, rvpiWidowOrphanControl,
    rvpiTabs, rvpiBiDiMode, rvpiOutlineLevel,
    rvpiCustom, rvpiCustom2, rvpiCustom3, rvpiCustom4, rvpiCustom5);

TRVParaInfoProperties = set of TRVParaInfoProperty;
```

Identifies properties of TParaInfo [448].

This is a type of the following properties:

- TRVStyleTemplate.ValidParaProperties [463];
- TParaInfo.ModifiedProperties [457].

Parameters of this type are used in the following methods:

- TParaInfos.FindSuchStyle, FindSuchStyleEx [425];
- TParaInfo.IsEqual [458].

```
const
  RVAllParaInfoProperties: TRVParaInfoProperties =
    [Low(TRVParaInfoProperty)..High(TRVParaInfoProperty)];
```

A constant, lists all TParaInfo properties.

1.8.29 TRVPrintingStep

Unit PtblRV

```
type
  TRVPrintingStep = (rvpsStarting, rvpsProceeding, rvpsFinished);
```

Parameter of this type is used in OnSendingToPrinter [496] and OnFormatting [494] events.

1.8.30 TRVRawByteString

Unit RVTypes.

Declaration for Delphi 2009 or newer:

```
type
  TRVRawByteString = type RawByteString;
```

Declaration for older versions of Delphi:

```
type
  TRVRawByteString = type String;
```

String of this type may contain arbitrary data, not necessary an ANSI text.

Usually, this type is used for strings containing item text. In this case, it contains "raw Unicode" for text items [88] associated with Unicode [446] text style, and ANSI string otherwise.

See also

- TRVAnsiString [623]

- TRVUnicodeString 647

1.8.31 TRVReaderStyleMode

Unit RVStyle;

```
type
  TRVReaderStyleMode =
    (rvrsUseSpecified, rvrsUseClosest, rvrsAddIfNeeded);
```

Defines modes for loading formatting from RTF.

Mode	Meaning
rvrsUseSpecified	All character/paragraph formatting in RTF will be ignored, all text/paragraphs will be formatted with the specified style.
rvrsUseClosest	RTF formatting will be mapped to the the most similar existing style. New styles will not be added.
rvrsAddIfNeeded	This mode provides the most closer look to the original RTF. RichView will try to use existing styles as much as possible, but if style with the desired formatting does not exist, RichView will create and use a new style.

See TRichView.RTFReadProperties.TextStyleMode 274 and ParaStyleMode 272.

1.8.32 TRVSaveFormat

Unit RVStyle;

```
type
  TRVSaveFormat =
  (rvsfText, rvsfHTML, rvsfRTF, rvsfRVF, rvsfDocX);
```

Identifies saving format.

This type is used in some events of RichView, such as OnSaveComponentToFile 256 and OnSaveItemToFile 261, and SavePicture 219 method.

1.8.33 TRVSaveOptions

Unit RVStyle;

```
type
  TRVSaveOption = (rvsoOverrideImages,
    rvsoFirstOnly, rvsoMiddleOnly, rvsoLastOnly,
    rvsoDefault0Style, rvsoNoHypertextImageBorders,
    rvsoImageSizes, rvsoForceNonTextCSS, rvsoUseCheckpointsNames,
    rvsoMarkersAsText, rvsoNoDefCSSStyle, rvsoInlineCSS,
    rvsoUseItemImageFileNames,
    rvsoXHTML, rvsoUTF8);
  TRVSaveOptions = set of TRVSaveOption;
```

Options for HTML export. Used for parameter of SaveHTML 216, SaveHTMLEx 217, SaveHTMLToStream 218, SaveHTMLToStreamEx 218.

Options for saving images, both for SaveHTML [216], SaveHTMLEx [217]

Option	Meaning
rvsoOverrideImages	If set, RichView can override existing images. If not set, RichView will generate unique names for images.
rvsoNoHypertextImageBorders	By default, browsers add borders around hypertext images. This option suppresses borders for *hotspots* [97] and *hot-pictures* [93].
rvsoImageSizes	If set, RichView saves widths and heights of images in HTML explicitly. This option provides more smooth (and sometimes faster) loading by browsers. This option does not affect pictures having defined *rvepImageWidth* and/or *rvepImageHeight* item extra integer properties [627]. Sizes are always saved for such images. You can override this setting for the specific item using *rveoNoHTMLImagesSize* item extra integer properties [627] .
rvsoUseItemImageFileNames	If included, images with defined (non-empty) file names will not be saved, but their file names will be written in HTML. If this option is included, OnHTMLSaveImage [241] and OnSaveImage2 [260] events have initial value of the Location parameter equal to the image file name. File names for items are defined via *rvespImageFileName* [630] string property [630].

Options for saving list markers, both for SaveHTML [216], SaveHTMLEx [217]

Option	Meaning
rvsoMarkersAsText	If included, paragraph lists [99] are saved as normal text or images. Usually, HTML files saved with this option look closer to the original document. If not included, lists are saved as and HTML tags.

If *rvsoMarkersAsText* is not included

Export using CSS (SaveHTMLEx [217]); CSS of lists are inserted directly in HTML:

- *rvlstBullet, rvlstUnicodeBullet* [476] – exported as bullets with default markers (disc, circle or square). FormatString [474] and Font [473] are ignored;
- *rvlstDecimal,rvlstLowerAlpha,rvlstUpperAlpha,rvlstLowerRoman,rvlstUpperRoman* [476] – exported as lists with the proper numbering. FormatString [474] and Font [473] are ignored;
- *rvlstPicture,rvlstImageList,rvlstImageListCounter* [476] – exported as a bullet with picture. It's possible to change default image saving by OnHTMLSaveImage [241];
- indents are saved, but not very accurately.

Export without CSS (SaveHTML [216]):

- *rvlstBullet, rvlstUnicodeBullet,rvlstPicture,rvlstImageList* [476] – exported as bullets with default markers (disc, circle or square). FormatString [474], Picture [479], ImageList [475] and Font [473] are ignored;
- *rvlstDecimal,rvlstLowerAlpha,rvlstUpperAlpha,rvlstLowerRoman,rvlstUpperRoman* [476] – exported as lists with the proper numbering. FormatString [474] and Font [473] are ignored;

- *rvlstImageListCounter* 476 – saved as a decimal numbering;
- indents are not saved.

If *rvsoMarkersAsText* is included

Markers are saved without special HTML keywords for lists (like <ol>, <ul>, <li>).

Font 473 and format strings 474 settings are respected.

Saving without CSS (SaveHTML 216): all indents are ignored.

Saving with CSS (SaveHTMLEx 217): LeftIndent 476 and MarkerIndent 478 are retained. FirstIndent 473 are retained for non-hanging markers only (MarkerIndent>=LeftIndent)..

Generally, HTML files saved with this option look closer to the original.

Other options, both for SaveHTML 216, SaveHTMLEx 217

Option	Meaning
rvsoUseCheckpointsNames	*Checkpoints* 30' names are used instead of checkpoint indices. Recommended.
rvsoXHTML	If included, XHTML is saved instead of HTML
rvsoUTF8	If included, Unicode (UTF-8) HTML file is saved. Highly recommended for multilingual documents.
rvsoDefault0Style	If set, no formatting will be saved for text of the 0-th style (TextStyles 404[0]). Browsers will use a default font for it. This option affects the appearance of all styles, because only attributes different from attributes of the 0th style are saved.

Options for SaveHTML 216

Option	Meaning
rvsoForceNonTextCSS	CSS is used for exporting non-text items. This option allows to retain colors of breaks and layout of tables saved with SaveHTML (without using CSS for text and paragraph styles). Note: SaveHTMLEx always uses CSS in this case.

Options for SaveHTMLEx 217

Option	Meaning
rvsoNoDefCSSStyle	By default, SaveHTMLEx assigns attributes of TextStyles 404[0] to <body> and <table>, and attributes of all other styles are saved relative to attributes of TextStyles[0]. If you include this option, TextStyles[0] will be treated like any other style. The same for ParaStyles 398[0]. It's not recommended to use this option: it generates much larger CSS and HTML. This option is useful if HTML generated by TRichView is used as a part of larger HTML, and you do not want to change

	properties of <body> and <table>.
rvsoInlineCSS	If included, CSS is inserted directly in <p>, and tags (instead of generating CSS table at the beginning of HTML). This option generates highly overbloated HTML. Not recommended.

Options for saving only part of document, both for SaveHTML[216], SaveHTMLEx[217]

These options are useful when generating HTML documents what will be inserted in larger HTML document.

Option	Meaning
rvsoFirstOnly	If set, RichView saves only opening part of HTML (opens <html> tag, saves full <head> tag, opens <body> tag)
rvsoMiddleOnly	If set, RichView saves only body of HTML document (content inside <body> tag)
rvsoLastOnly	If set, RichView saves closing part of HTML document (closes <body> and <html> tags)

If none of these three options specified, full HTML document is saved.

You cannot set two or all of these flags at the same time (nothing will be saved in such case).

1.8.34 TRVSearchOptions

Unit RVScroll.

```
type
  TRVSearchOption = (rvsroMatchCase, rvsroDown, rvsroWholeWord, rvsroFromStart,
    rvsroMultiItem, rvsroSmartStart);
  TRVSearchOptions = set of TRVSearchOption;
```

Options for searching in RichView, see TCustomRichView.SearchText[222].

Value	Meaning
rvsroMatchCase	If included, a character case is taken into account when comparing strings.
rvsroDown	If included, the search is performed to the end of the document. If not included, it is performed to the top of the document.
rvsroWholeWord	If included, the searched string matches only whole words.
rvsroFromStart	If included, the search starts from the beginning/end of the document. If not included, it starts from the end of selection or the first/last visible item.
rvsroMultiItem	If included, the search can match substrings of several text items[88]. If not included, the text is searched in each text item separately. For example, if the document contains the text **He*llo***, the substring 'Hello' can be found

	only if this option is included, because 'He' and 'llo' belong to different items.
rvsroSmartStart	This option is used only when *rvsroFromStart* is not included. If included, and a document fragment is selected, then: ▪ when searching down, the search starts from the second selected character; ▪ when searching up, the search starts from the last but one selected character (selected characters are counted from the top of the document, the selection direction is ignored)

1.8.35 TRVSeqType

Unit RVStyle.

```
type
  TRVSeqType = (rvseqDecimal, rvseqLowerAlpha,
    rvseqUpperAlpha, rvseqLowerRoman, rvseqUpperRoman);
```

Numbering type for sequences [102], endnotes [103], footnotes [104], sidenotes [105].

Type	Meaning
rvseqDecimal	1, 2, 3,...
rvseqLowerAlpha	a, b, c, ..., z, aa, ab, ...
rvseqUpperAlpha	A, B, C, ..., Z, AA, AB, ...
rvseqLowerRoman	i, ii, iii, iv, ...
rvseqUpperRoman	I, II, III, IV, ...

This is a type of the following properties:

- TRVStyle.EndnoteNumbering [389];
- TRVStyle.FootnoteNumbering [389];
- TRVStyle.SidenoteNumbering [389];
- TRVSeqItemInfo.NumberType [584].

1.8.36 TRVStrProperty

Unit RichView.

Identifies a property of TRichView [128] of a string type.

```
type
  TRVStrProperty = (rvspDPTitle, rvspDPAuthor, rvspDPComments);
```

Value	Corresponding property of TRichView
rvspDPTitle	DocParameters [140].Title [280]
rvspDPAuthor	DocParameters.Author [277]
rvspDPComments	DocParameters.Comments [278]

A parameter of this type is used in TRichViewEdit[282].SetStrPropertyEd[337].

See also:

- TRVFloatProperty[632];
- TRVIntProperty[635].

1.8.37 TRVStyleLength

Unit RVStyle;

```
type
  TRVStyleLength = type Integer;
```

This is a type of values measured in TRVStyleUnits[644] (i.e. either twips or screen pixels).

See also:

Units of measurement in TRichView[74]

1.8.38 TRVStyleTemplateId

Unit RVStyle;

```
type
  TRVStyleTemplateId = type Integer;
```

Values of this type identify items in TRVStyleTemplateCollection[427],

This is a type of the following properties:

- Id[461] of TRVStyleTemplate[460] (unique style template identifier);
- ParentId[463] of TRVStyleTemplate[460] (a reference to the parent style template);
- NextId[462] of TRVStyleTemplate[460] (a reference to the style template for following paragraphs);
- StyleTemplateId[432] of text and paragraph attributes (styles), linking them to a style template.
- ParaStyleTemplateId[439] of text attributes (styles), linking them to a style template.

Valid identifiers are positive values. When a value of this type is used as a reference, the value -1 means "no style template". The value 0 is reserved.

1.8.39 TRVStyleTemplateName

Unit RVStyle;

```
type
  TRVStyleTemplateName = type String;
```

This is a type of the following properties:

- Name of TRVStyleTemplate[460] (unique style template name)

1.8.40 TRVStyleUnits

Unit RVStyle.

```
type
  TRVStyleUnits = (rvstuPixels, rvstuTwips);
```

Value	Meaning
rvstuPixels	Screen pixels (see below)
rvstuTwips	Twips. 1 twip = 1/20 of point = 1/1440 inch

If RichViewPixelsPerInch[655] is positive, it is used for calculation of number of screen pixels in one inch. Otherwise, Screen.PixelsPerInch is used. In the latter case, values in *rvstuPixels* depend on the screen

resolution and cannot be used in WYSIWYG editors.

Values measured in TRVUnits have type TRVStyleLength[644].

This is a type of TRVStyle[385].Units[404].

See also:

Units of measurement in TRichView[74]

1.8.41 TRVTableBorderStyle

Unit RVTable;

```
type
  TRVTableBorderStyle =
    (rvtbRaised, rvtbLowered, rvtbColor,
     rvtbRaisedColor, rvtbLoweredColor);
```

Defines style of border in table. This is a type of BorderStyle[533] and CellBorderStyle[535] properties of table.

Value	Meaning
rvtbRaised	Raised 3d border, with default colors.
rvtbLowered	Sunken 3d border, with default colors.
rvtbColor	Flat border of BorderColor[532]/CellBorderColor[534] color.
rvtbRaisedColor	Raised 3d border: ▪ left and top sides: BorderLightColor[532]/CellBorderLightColor[534] ; ▪ bottom and right sides: BorderColor[532]/CellBorderColor[534]).
rvtbLoweredColor	Sunken 3d border: ▪ left and top sides: BorderColor[532]/CellBorderColor[534]); ▪ bottom and right sides: BorderLightColor[532]/ CellBorderLightColor[534]).

See also

- Colors and layout of tables[114]

1.8.42 TRVTag

Unit RVStyle;

```
type
  TRVTag = type String;
```

This is the type of item tags – additional value associated with any item[85] in TRichView[128] document, and with any table cell[574].

See also:

- Tags[34]

1.8.43 TRVTextDrawStates

Unit RVStyle;

```
type
  TRVTextDrawState = (rvtsSelected, rvtsHover,
    rvtsItemStart, rvtsItemEnd,
```

```
    rvtsDrawItemStart, rvtsDrawItemEnd,
    rvtsControlFocused, rvtsSpecialCharacters);
  TRVTextDrawStates = set of TRVTextDrawState;
```

States of text, used for drawing.

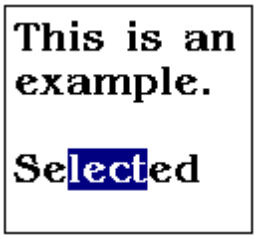
See:

- TRVStyle.OnDrawStyleText [415], OnDrawTextBack [416];
- TFontInfo.Draw [447]

State	Meaning
rvtsSelected	text is selected (for copying to the Clipboard)
rvtsHover	[hyper]text is below a mouse pointer
rvtsItemStart	this call of event draws text item from the beginning
rvtsItemEnd	this call of event finishes drawing of text item
*rvtsDrawItemStart**	this call of event draws part of text item (called "drawing item") from the beginning
*rvtsDrawItemEnd**	this call of event finishes drawing of part of text item.
rvtsControlFocused	TRichView has input focus
rvtsSpecialCharacters	*rvoShowSpecialCharacters* is included in TRichView.Options [146].

* old versions of the components might draw selection as a part of a string belonging to a drawing item; the new version always draws the whole drawing item, setting the proper clipping region before drawing, so these states can be ignored.

For example



(there are two items: 'This is an example.' and 'Selected',

and three "drawing items": 'This is an', 'example.' and 'Selected'

Calls of TRVStyle.OnDrawStyleText [415] will be:

'This is an' with [*rvtsItemStart,rvtsDrawItemStart,rvtsDrawItemEnd*]

'example.' with [*rvtsItemEnd, rvtsDrawItemStart, rvtsDrawItemEnd*]

'Selected' with [*rvtsItemStart, rvtsItemEnd, rvtsDrawItemStart, rvtsDrawItemEnd*]

'Selected' with [*rvtsItemStart, rvtsItemEnd, rvtsDrawItemStart, rvtsDrawItemEnd, rvtsSelected*] (a clipping region is set so that only 'lect' is visible).

(Note: in older versions of the components, the last cal was 'lect' with [*rvtsSelected*]).

1.8.44 TRVUndoType

Unit RVEdit

```
type
TRVUndoType = (
    rvutNone, rvutDelete, rvutInsert, rvutPara, rvutMiscTyping,
    rvutInsertPageBreak, rvutRemovePageBreak,
    rvutTyping, rvutTag, rvutStyleNo,
    rvutAddCheckpoint, rvutRemoveCheckpoint, rvutModifyCheckpoint,
    rvutModifyItem, rvutToHypertext,  rvutFromHypertext,
    rvutTextFlow, rvutList, rvutProperty, rvutLayout,
    rvutBackground, rvutStyleTemplate, rvutSubDoc,
    rvutResize, rvutTableCell,   rvutTableRow,
    rvutTable, rvutTableInsertRows, rvutTableInsertCols,
    rvutTableDeleteRows, rvutTableDeleteCols,
    rvutTableMerge, rvutTableSplit,
    rvutCustom);
```

The values (except for *rvutNone* and *rvutCustom*) identify types of standard undo operations. They are useful if you wish to show (in a menu or a hint) which operation is about to be undone.

There are no hard-coded names for these operations, because they may be application specific. For example, *rvutTag* (changing item's *tag* 34) can have special meaning for an application (editing a hyperlink or another item property).

The demo **Demos\Delphi\Editors\Editor 1** has suggested names of actions in **RVUndoStr.pas**.

There are two special undo types:

Type	Meaning
rvutNone	No undo/redo available (disable an undo or a redo menu item or button)
rvutCustom	The undo action has a special name (see BeginUndoCustomGroup 302).

1.8.45 TRVUnicodeString

Unit RVTypes.

Declaration for Delphi 2009 or newer:

```
type
  TRVUnicodeString = type UnicodeString;
```

Declaration for older versions of Delphi:

```
type
  TRVUnicodeString = type WideString;
```

Strings of this type contain Unicode text (UTF-16).

See also

- TRVAnsiString 623
- TRVRawByteString 638

1.8.46 TRVUnits

Unit RVStyle.

Units of measurement.

```
type
  TRVUnits = (rvuInches, rvuCentimeters, rvuMillimeters, rvuPicas, rvuPixels, rvuPoints);
```

Value	Meaning
rvuInches	Inches. 1 inch = 2.54 cm.
rvuCentimeters	Centimeters.
rvuMillimeters	Millimeters.
rvuPicas	Picas (used in typography). 1 pica = 1/6 inch = 12 points.
rvuPixels	Screen pixels (see below).
rvuPoints	Points (used in typography). 1 point = 1/72 inch.

If RichViewPixelsPerInch[655] is positive, it is used for calculation of number of screen pixels in one inch. Otherwise, Screen.PixelsPerInch is used. In the latter case, values in *rvuPixels* depend on the screen resolution and cannot be used in WYSIWYG editors.

This is a type of the following properties:

- TRichView[128].DocParameters[140].Units[280].
- TRVPrint[479].Units[487]

Values measured in TRVUnits have the type TRVLength[636].

See also:

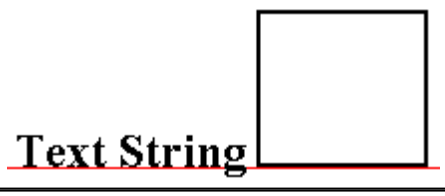
Units of measurement in TRichView[74]

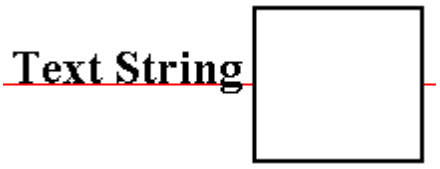
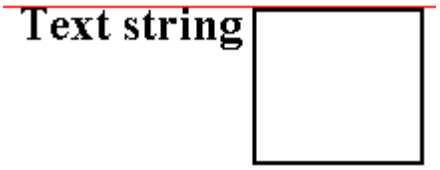
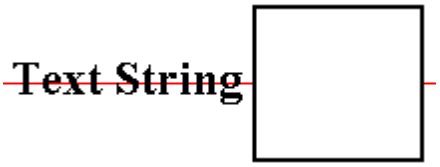
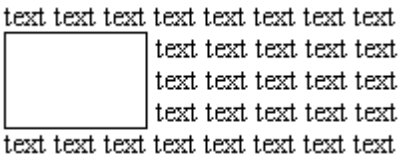
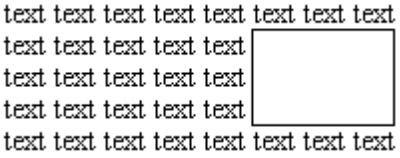
1.8.47 TRVVAlign

Unit RVStyle;

```
type
  TRVVAlign = (
    rvvaBaseline,  // bottom of picture -> baseline
    rvvaMiddle,    // center of picture -> baseline
    rvvaAbsTop,    // top of picture    -> top of line
    rvvaAbsBottom, // bottom of picture -> bottom of line
    rvvaAbsMiddle, // center of picture -> center of line
    rvvaLeft,      // align to left side
    rvvaRight      // align to right side
    );
```

Vertical align of pictures and controls in RichView

Align	Scheme
rvvaBaseline	

rvvaMiddle	
rvvaAbsTop	
rvvaAbsBottom	
rvvaAbsMiddle	
rvvaLeft	
rvvaRight	

Note: non-standard values of HTML align attributes (abstop, absbottom, absmiddle) are used for export of rvaAbs*** values. Microsoft Internet Explorer and Mozilla understand them, but not Opera.

This property can be applied to:

- pictures 90 and *hot-pictures* 93
- *bullets* 96 and *hotspots* 97
- controls 94
- label items 101
- numbered sequences 102

See also methods of TCustomRichView 128:

- GetItemVAlign 196,
- SetItemVAlign 233.

See also properties of TCustomRichView 128:

- ClearLeft 138,
- ClearRight 138.

See also methods of TCustomRichViewEdit 282:

- GetCurrentItemVAlign 314,
- SetItemVAlignEd 353,
- SetCurrentItemVAlign 353,
- ClearTextFlow 305.

See also properties of TRVStyle 385:

- FloatingLineColor 390

1.9 Global Constants and Variables

Global Variables and Typed Constants

Editing, Caret

- RichViewEditCaretWidth 652: Integer – the caret width;
- RichViewEditCaretHeightExtra 652: Integer, allows making the caret shorter or longer;
- RichViewEditMaxCaretHeight 652: Integer – maximal possible caret height;
- RichViewEditMinCaretHeight 652: Integer – minimal possible caret height.

Editing, Other

- RichViewEditDefaultProportionalResize 654: Boolean, specifies whether images are resized proportionally with the mouse;
- RichViewEditEnterAllowsEmptyMarkeredLines 654: Boolean, enables or disables autodeletion of list marker from new empty lines;
- RichViewUnicodeInput 657: TRichViewUnicodeInput, specifies how types Unicode characters are processed (in non-Unicode versions of Delphi);
- RichViewTableAutoAddRow 657: Boolean, allows or disallows adding new table rows using Tab key.

Saving and Loading

- RichViewSavePInHTML 652: Boolean, instructs always using <p> for saving in HTML;
- RichViewSaveDivInHTMLEx 652: Boolean, instructs always using for saving in HTML;
- RichViewSavePageBreaksInText 652: Boolean, allows or disallows saving page breaking characters in plain text;
- RichViewDoNotCheckRVFStyleRefs 653: Boolean, allows turning off checking for invalid style indices when loading RVF;
- RichViewResetStandardFlag 656: Boolean, specifies whether to reset Standard 431 property to False when inserting RVF;
- RichViewPixelsPerInch 655: Integer, allows overriding the screen resolution;
- RichViewAutoAssignNormalStyleTemplate 653: Boolean, specifies whether to assign "Normal" style template when loading unstyled paragraphs from RVF and RTF.
- RichViewTableDefaultRTFAutofit 657: Boolean, allows saving tables as fit-by-content tables in RTF and DocX.
- RichViewSaveHyperlinksInDocXAsFields 656 defines how hyperlinks are saved in DocX

Tables

- RichViewTableAutoAddRow 657: Boolean, allows or disallows adding new table rows using Tab key;
- RichViewAllowCopyTableCells 652: Boolean allows preventing copying multicell selection in a table to the Clipboard.
- RichViewTableDefaultRTFAutofit 657: Boolean, allows saving tables as fit-by-content tables in RTF and DocX.

Formatting and Displaying

- RichViewJustifyBeforeLineBreak 654:Boolean, specifies whether lines before line breaks are justified (expanded);
- RichViewStringFormatMethod 656:Boolean, allows activating an alternative text measuring method;
- RichViewShowGhostSpaces 656: Boolean, allows showing space characters "hidden" between lines.
- RVVisibleSpecialCharacters 665: TRVSpecialCharacters specifies which special characters can be shown.
- RVThumbnailMaker 664: TRVThumbnailMaker 615, provides access to an object used to create scaled images;

- RichViewMaxThumbnailCount 655: Integer – maximal count of scaled images stored in TRichView control.

Printing

- RichViewPixelsPerInch 655: Integer, allows overriding the screen resolution;
- RichViewAlternativePicPrint 652: Boolean, allows using an alternative picture printing method.

Live spelling check

- RichViewApostropheInWord 653: Boolean, specifies how apostrophe characters are processed when finding word boundaries.

Classes

- RVThumbnailMaker 664: TRVThumbnailMaker 615, provides access to an object used to create scaled images;
- RVGraphicHandler 662: TRVGraphicHandler 613, provides access to an object used to manage graphic classes.

Other

- RichViewCompareStyleNames 653: Boolean, specifies whether StyleName 432 property is used when comparing (and searching for) text, paragraph or list styles.

Global Functional Variables

- RVIsCustomURL 662: TCustomRVIsURLFunction, allows defining your own URL prefixes.

Constants

- **rvs*** 663 constants 663: item types;
- **rveipc*** 658 constants 658: integer properties of items;
- **rvespc*** 661 constants 658: string properties of items;
- Constants and variables related to Clipboard 651.

1.9.1 Clipboard Related Constants and Variables

Unit RVStr.

Clipboard Format Names

Names of the Clipboard formats for copying RVF 59, RTF, URL, HTML.

Syntax

```
const
  RVFormatName = 'RichView Format';
  RTFormatName = 'Rich Text Format';
  URLFormatName = 'UniformResourceLocator';
  HTMLFormatName = 'HTML Format';
```

Clipboard Format Identifiers

```
var CFRV_RVF, CFRV_RTF, CFRV_HTML, CFRV_URL: Word;
  ...
  CFRV_RVF := RegisterClipboardFormat(RVFormatName);
  CFRV_RTF := RegisterClipboardFormat(RTFormatName);
  CFRV_URL := RegisterClipboardFormat(URLFormatName);
  CFRV_HTML := RegisterClipboardFormat(HTMLFormatName);
```

1.9.2 Constants Related to HTML and Text

Unit CRVData.

Syntax

```
const
  RichViewSavePInHTML: Boolean = False;
  RichViewSaveDivInHTMLEx: Boolean = False;
  RichViewSavePageBreaksInText: Boolean = False;
```

Set **RichViewSavePInHTML** to *True* is you want SaveHTML[216] to write <p> instead of in HTML. This variable does not affect SaveHTMLEx[217]. This setting changes appearance of resulting HTML documents (usually browsers add additional spacing between <p>).

Set **RichViewSaveDivInHTMLEx** to *True* is you want SaveHTMLEx[217] to write instead of <p> in HTML. This variable does not affect SaveHTML[216]. This setting does not change appearance of resulting HTML documents (because all <p> or visual attributes are defined in CSS).

Set **RichViewSavePageBreaksInText** to *True* if you want to save explicit page breaks as #$0C characters. This typed constant affects all plain text export methods.

1.9.3 Constants Related to the Caret

These typed constants allow to control the caret appearance in RichViewEdit[282].

Unit RVERVData.

Syntax

```
const RichViewEditCaretWidth: Integer = 1;
      RichViewEditCaretHeightExtra: Integer = 0;
      RichViewEditMaxCaretHeight: Integer = 1000;
      RichViewEditMinCaretHeight: Integer = 0;
```

RichViewEditCaretWidth – width of the caret.

Value of **RichViewEditCaretHeightExtra** is added to the top and the bottom of the caret (negative values make the caret shorter, positive values make it longer).

RichViewEditMaxCaretHeight – maximal possible caret height.

RichViewEditMinCaretHeight – minimal possible caret height.

1.9.4 RichViewAllowCopyTableCells

Unit CRVData.

Syntax

```
const
  RichViewAllowCopyTableCells: Boolean = True;
```

Set to *False* to disallow copying multicell selection (it will be copied as empty RVF[59]).

1.9.5 RichViewAlternativePicPrint

Unit RVFuncs.

Syntax

```
const
  RichViewAlternativePicPrint: Boolean = False;
```

If set to *True*, alternative picture printing procedure is used. The alternative procedure is less reliable, but does not depend on the screen color resolution. It may be useful if your application work on server in 16 or 256 color mode.

1.9.6 RichViewApostropheInWord

Unit RVThread;

Syntax

```
var
  RichViewApostropheInWord: Boolean = True;
```

Specifies how apostrophe characters ("" and #146) are processed in live spelling.

If *True*: apostrophes are allowed in middle of words, even if they are included in Delimiters[140] property. For example, "doesn't" is processed as one word instead of "doesn" and "t".

If *False*: apostrophes are processed like all other delimiters. For example, when checking French text with Addict3, it is recommended to include "" in Delimiters[140] and assign RichViewApostropheInWord=*False*.

Note: apostrophe is not included in Delimiters[140] by default.

1.9.7 RichViewAutoAssignNormalStyleTemplate

Unit RVStyle.

Syntax

```
const
  RichViewAutoAssignNormalStyleTemplate: Boolean = True;
```

(introduced in version 14)

This constant controls how unstyled paragraphs (i.e. paragraphs not linked[432] to any style template[460]) are loaded from RTF and RVF files/streams, if style templates are used[158].

If *True*, "Normal" style template[428] (if it exists) is assigned to paragraph styles[456] of the loaded document.

If *False*, -1 is assigned to paragraph styles of the loaded document.

This constant affects the both loading and insertion. See also TRichView.StyleTemplateInsertMode[156] property.

See also:

- Styles and style templates[40].

1.9.8 RichViewCompareStyleNames

Unit RVStyle.

Syntax

```
const
  RichViewCompareStyleNames: Boolean = False;
```

If set to *True*, methods for searching and comparing styles take StyleName[432] property into account.

1.9.9 RichViewDoNotCheckRVFStyleRefs

Unit CRVData.

Syntax

```
const
  RichViewDoNotCheckRVFStyleRefs: Boolean = False;
```

(introduced in version 10)

If set to *True*, TRichView ignore invalid style references when loading RVF documents [59].

It is useful only if you do not store collections of styles in documents and want to delete unused styles in multiple documents, see TRVDeleteUnusedStylesData [612].

1.9.10 RichViewDoNotMergeNumbering

Unit RVStyle.

Syntax

```
const
  RichViewDoNotMergeNumbering: Boolean = False;
```

RichViewDoNotMergeNumbering affects inserting RVF [59]. By default, when inserting RVF files containing numbered lists, TRichView tries to map the inserted list styles to the existing list styles: if the inserted list style A has the same properties (including a *list identifier*) as the existing style B, then A is mapped to B; so inserted numbered lists of the style A continue existing numbered lists of the style B. *List identifier* is a random number assigned to any list style on creation. Because of list identifiers, numbered lists that are copy-pasted from the same document continue the existing numbering (because they have the same identifiers), while numbered lists from another document are inserted as new lists.

Assign *True* to **RichViewDoNotMergeNumbering** to modify the standard behavior. In this mode, all numbered lists from RVF are inserted as new lists, even if the document has list styles with the same properties and identifiers.

This property is useful when generating a big document from several small documents.

1.9.11 RichViewEditDefaultProportionalResize

Unit RVERVData.

Syntax

```
const
  RichViewEditDefaultProportionalResize: Boolean = True;
```

By default, dragging corner handles resizes picture proportionally (like in MS Word).

Set to *False* to enable free resizing using the corner handles (use **Shift** key to resize proportionally in this mode)

1.9.12 RichViewEditEnterAllowsEmptyMarkeredLines

Unit RVERVData.

Syntax

```
const
  RichViewEditEnterAllowsEmptyMarkeredLines: Boolean = False;
```

Set to *True* if you do not want to autoremove bullets & numbering from blank lines when the user presses **Enter**.

1.9.13 RichViewJustifyBeforeLineBreak

Unit CRVFData.

Syntax

```
const
  RichViewJustifyBeforeLineBreak: Boolean = False;
```

(introduced in version 15)

If *False*, the last line before a line break (added with Shift+Enter) is not justified by expanding space characters.

Set to *True* to justify such lines.

1.9.14 RichViewMaxThumbnailCount

Unit CRVData.

Syntax

```
const RichViewMaxThumbnailCount = 20;
```

(introduced in version 15)

Specifies the maximal count of scaled images stored in TCustomRichView[128].

If RichViewMaxThumbnailCount=-1, all scaled images are stored.

See also:

- Smooth image scaling[57].

1.9.15 RichViewPixelsPerInch

Number of pixels in one screen inch.

Unit RVFuncs.

Syntax

```
const
  RichViewPixelsPerInch : Integer = 0;
```

If 0 (default), RichView uses the screen resolution (Screen.PixelsPerInch).

If non zero, this value is used.

This value is used when converting pixels on the screen to mm, inches, twips, pixels on the printer, and vice versa.

The most important applications are:

- printing;
- RTF export&import;
- calculating sizes of objects when they are measured in twips[74].

This constant does not affect text (TRVStyle[385].TextStyles[404].PixelsPerInch[419] property is used for text, it has the same meaning). This constant affects pictures and tables.

The main possible values: 96 (small font mode), 120 (large font mode).

Assigning positive value this constant makes printing and RTF export independent of the screen resolution (small font/large fonts mode).

If you use TRVRuler component (from RichViewActions) and if you assign a positive value to RichViewPixelsPerInch, assign the same value to RVRuler1.ScreenRes property,

1.9.16 RichViewResetStandardFlag

Unit RVStyle.

Syntax

```
const
  RichViewResetStandardFlag: Boolean = True;
```

If *True*, Standard[431] properties of styles which were added from inserted RVF[59] will be reset to *False* (it only affects RVF insertion, not loading).

If this flag is *True*, new styles having Standard[431]=*True* may appear only after calling LoadRVF[211] or LoadRVFFromStream[211].

1.9.17 RichViewSaveHyperlinksInDocXAsFields

Unit CRVData.

Syntax

```
const
    RichViewSaveHyperlinksInDocXAsFields = False;
```

This constant switches between two methods of hyperlinks saving in DocX files.

If *False*:

SaveDocX[215] saves hyperlinks as <w:hyperlink>. This way of saving is more compatible, it must be understood by all DocX readers.

Minus: MS Word checks hyperlink targets more strictly, if it contains an invalid URL, it may treat the whole file as erroneous.

If *True*:

SaveDocX saves hyperlinks as HYPERLINK field, similar to RTF.

1.9.18 RichViewShowGhostSpaces

Unit CRVFData.

Syntax

```
const
  RichViewShowGhostSpaces: Boolean = False;
```

(introduced in version 10)

Set to *True* if you want to show space characters at the end of lines when *rvoShowSpecialCharacters* is included in TCustomRichView.Options[146]. Caret cannot be placed after these spaces.

1.9.19 RichViewStringFormatMethod

Unit CRVFData.

Syntax

```
type
  TRVStringFormatMethod =
    (rvsfmAuto, rvsfmEngine, rvsfmTRichView);
const
  RichViewStringFormatMethod: TRVStringFormatMethod = rvsfmAuto;
```

This variable specifies formatting methods used in all TRichView and ScaleRichView components. The main question is the method of calculation of the count of characters in the string that fit to the specified width.

Value	Meaning
rvsfmAuto	The components tries to choose the fastest method
rvsfmEngine	The component uses the function provided by API
rvsfmTRichView	The component uses its own method (a binary search on string widths).

About the "auto" method

If TextEngine[403]=*rvteUniscribe*, or in ScaleRichView, the components choose the TRichView method only if the string length is much greater than an estimated count of character fitting to the specified width.

Otherewise, the components always choose the TRichView method for

1.9.20 RichViewTableAutoAddRow

Unit RVTable.

Syntax

```
const
  RichViewTableAutoAddRow: Boolean = True;
```

(introduced in version 10)

Set to *False* to disable auto insertion of new table row when user presses **Tab** key in the last table[98] cell.

The row auto insertion works only in TRichViewEdit[282] and TDBRichViewEdit[377].

1.9.21 RichViewTableDefaultRTFAutofit

Unit RVTable.

Syntax

```
const
  RichViewTableDefaultRTFAutofit: Boolean = False;
```

Assigning *True* to this variable works like inclusion of *rvtoRTFAllowAutofit* in Options[540] properties of all tables[98].

1.9.22 RichViewUnicodeInput

Unit RVEdit.

Syntax

```
type
  TRichViewUnicodeInput = (rvuiStandard, rvuiAlternative);
const RichViewUnicodeInput: TRichViewUnicodeInput = rvuiStandard;
```

This constant is ignored for Delphi/C++Builder 2009 or newer (they provide true Unicode text input).

RichViewEdit uses two methods to process typing of unicode[64] text:

- "Standard" method processes WM_CHAR and converts typed characters to Unicode.
- "Alternative" method uses ToUnicode procedure. It cannot process "dead keys" correctly, so it should not be used for keyboard layouts having "dead keys".

If RichViewUnicodeInput=*rvuiStandard*, the "alternative" method is used only for the following languages:

- languages without their own code page (so it's impossible to use WM_CHAR);
- Kazakh, Tatar, Mongolian, Azerbaijani (cyrillic&latin), Uzbek, Kyrgyz, Serbian (these languages include some characters that do not belong to code page).

Generally, it works very well. But for some "exotic" keyboard layouts the "standard" method does not provide completely correct input. Set this variable to *rvuiAlternative* to always use the "alternative" method. Do it only as an option, if you have troubles with the specific keyboard layout.

1.9.23 rveipcXXX constants

These constants identify integer properties of TRichView items[85].

They are used in the following methods:

- TRichView.GetExtraItemIntPropertyEx[193]
- TRichView.SetExtraItemIntPropertyEx[230]
- TRichViewEdit.SetItemExtraIntPropertyExEd[350]
- TRichViewEdit.GetCurrentItemExtraIntPropertyEx[312]
- TRichViewEdit.SetCurrentItemExtraIntPropertyEx[345]

Properties of Bullets[96] and Hotspots[97]

These properties are defined in RVItem unit.

These properties provides an alternative to the following methods:

- TRichView: SetBulletInfo[227], GetBulletInfo[184], GetHotspotInfo[189], SetHotspotInfo[229];
- TRichViewEdit: SetBulletInfoEd[339], SetCurrentBulletInfo[342], GetCurrentBulletInfo[308], SetHotspotInfoEd[349], SetCurrentHotspotInfo[345], GetCurrentHotspotInfo[310].

Constant	Value	Property
rveipcImageIndex	200	Image index (for *bullets* and *hotspots*)
rveipcHotImageIndex	201	Image index below the mouse pointer (for *hotspots*)

Properties of Breaks[93]

These properties are defined in RVItem unit.

These properties provides an alternative to the following methods:

- TRichView: SetBreakInfo[226], GetBreakInfo[183];
- TRichViewEdit: SetBreakInfoEd[339], SetCurrentBreakInfo[342], GetCurrentBreakInfo[308].

Constant	Value	Property
rveipcBreakWidth	200	Line width
rveipcBreakStyle	201	Line style*
rveipcBreakColor	202	Line color

* a property is type-casted to Integer

Properties of Label items [101]

These properties are defined in RVLabelItem unit.

They identify properties of label items [101] and items of all types inherited from them (numbered sequences [102], footnotes [104], endnotes [103], references to notes [108]).

Constant	Value	Property
rveipcProtectTextStyleNo	200	ProtectTextStyleNo [581]*
rveipcRemoveInternalLeading	201	RemoveInternalLeading [581]*
rveipcCursor	202	Cursor [580]
rveipcMinWidth	203	MinWidth [580]
rveipcAlignment	204	Alignment [580]*

* a property is type-casted to Integer

Properties of Numbered sequences [102]

These properties are defined in RVSeqItem unit.

They identify properties of numbered sequences [102] and items of all types inherited from them (footnotes [104], endnotes [103]).

Constant	Value	Property
rveipcSeqStartFrom	300	StartFrom [585]
rveipcSeqReset	301	Reset [584]*
rveipcSeqNumberType	302	NumberType [584]*,**

* a property is type-casted to Integer

** read-only for footnotes [104], endnotes [103], sidenotes [105] and text box items [107].

Properties of Sidenotes [105]

Position properties

These properties are defined in RVFloatingPos unit.

They identify subproperties of BoxPosition [592] property of sidenotes [105] and text box items [107].

Constant	Value	Property
rveipcFloatHorizontalAnchor	500	HorizontalAnchor [596]*
rveipcFloatHorizontalPositionKind	501	HorizontalPositionKind [597]*
rveipcFloatHorizontalOffset	502	HorizontalOffset [597]
rveipcFloatHorizontalAlign	503	HorizontalAlignment [597]*
rveipcFloatVerticalAnchor	504	VerticalAnchor [599]*

rveipcFloatVerticalPositionKind	505	VerticalPositionKind 600*
rveipcFloatVerticalOffset	506	VerticalOffset 600
rveipcFloatVerticalAlign	507	VerticalAlignment 600*
rveipcFloatRelativeToCell	508	RelativeToCell 599*
rveipcFloatPositionInText	509	PositionInText 599*

* a property is type-casted to Integer

Other properties

These properties are defined in RVFloatingBox unit.

They identify subproperties of BoxProperties 592 property of sidenotes 105 and text box items 107.

Constant	Value	Property
rveipcBoxWidth	600	Width 595
rveipcBoxHeight	601	Height 594
rveipcBoxWidthType	602	WidthType 595*
rveipcBoxHeightType	603	HeightType 594*
rveipcBoxBorderColor	604	Border 594.Color 468
rveipcBoxBorderWidth	605	Border 594.Width 468
rveipcBoxBorderInternalWidth	606	Border 594.InternalWidth 468
rveipcBoxBorderStyle	607	Border 594.Style 468*
rveipcBoxBorderVisibleBorders_Left	608	Border 594.VisibleBorders 468.Left*
rveipcBoxBorderVisibleBorders_Top	609	Border 594.VisibleBorders 468.Top*
rveipcBoxBorderVisibleBorders_Right	610	Border 594.VisibleBorders 468 .Right*
rveipcBoxBorderVisibleBorders_Bottom	611	Border 594.VisibleBorders 468 .Bottom*
rveipcBoxBorderBorderOffsets_Left	612	Border 594.BorderOffsets 466.Left
rveipcBoxBorderBorderOffsets_Top	613	Border 594.BorderOffsets 466.Top
rveipcBoxBorderBorderOffsets_Right	614	Border 594.BorderOffsets 466.Right
rveipcBoxBorderBorderOffsets_Bottom	615	Border 594.BorderOffsets 466 .Bottom
rveipcBoxColor	616	Background 594.Color 467
rveipcBoxPadding_Left	617	Background 594.BorderOffsets 466 .Left
rveipcBoxPadding_Top	618	Background 594.BorderOffsets 466 .Top

rveipcBoxPadding_Right	619	Background[594].BorderOffsets[466].Right
rveipcBoxPadding_Bottom	620	Background[594].BorderOffsets[466].Bottom
rveipcBoxVAlign	621	VAlign[595]*
rveipcBoxOpacity	622	Background[594].Opacity[467]

* a property is type-casted to Integer

Properties of "Page number" fields[109]

These properties are defined in RVFieldItems unit.

Constant	Value	Property
rveipcPageNumberType	600	NumberType[605]*

* a property is type-casted to Integer

See also

- rvespc*** constants[661]

1.9.24 rvespcXXX constants

These constants identify string properties of TRichView items[85].

They are used in the following methods:

- TRichView.GetExtraItemStrPropertyEx[193]
- TRichView.SetExtraItemStrPropertyEx[231]
- TRichViewEdit.SetItemExtraStrPropertyExEd[351]
- TRichViewEdit.GetCurrentItemExtraStrPropertyEx[313]
- TRichViewEdit.SetCurrentItemExtraStrPropertyEx[346]

Properties of Label items[101]

These properties are defined in RVLabelItem unit.

They identify properties of label items[101] and items of all types inherited from them (numbered sequences[102], footnotes[104], endnotes[103], sidenotes[105], references to notes[108], text boxes[107], page numbers[109], page counts[110]).

Constant	Value	Property
rvespcText	200	Text[581]*

* read-only for numbered sequences[102], footnotes[104], endnotes[103], sidenotes[105], references to notes[108]; ignored for text boxes[107]

Properties of Numbered sequences[102]

These properties are defined in RVSeqItem unit.

Constant	Value	Property
rvespcSeqName	300	SeqName[585]

rvespcSeqFormatString	301	FormatString 584

See also

- rveipc*** constants 658

1.9.25 RVGraphicHandler

Unit RVGrHandler.

Syntax

```
var RVGraphicHandler: TRVGraphicHandler 613;
```

(introduced in version 15)

This variable returns an object used by TRichView components to manage graphic classes.

1.9.26 RVIsCustomURL

Unit RVFuncs.

Syntax

```
type
  TCustomRVIsURLFunction = function (const Word: String): Boolean;
const
  RVIsCustomURL: TCustomRVIsURLFunction = nil;
```

You can assign your function to this variable. This function must return *True* if **Word** parameter contains URL. For example, you can create function returning *True* if **Word** is started from 'icq:'. It's not necessary to detect standard URL types here, detect only your own types of URLs.

This procedure is called by RVIsURL 619 function.

See demos:

- Demos\Delphi\Assorted\Hypertext\URLs\
- Demos\CBuilder\Assorted\Hypertext\URLs\

1.9.27 RVParagraphMarks

Unit RVStyle.

Syntax

```
type
  TRVParagraphMarks = (rvpmStandard, rvpmArrows);
const
  RVParagraphMarks: TRVParagraphMarks = rvpmStandard;
```

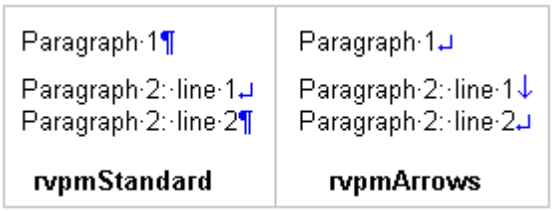
(introduced in version 13)

Defines a set of symbols used to display ends of paragraphs and lines.

These symbols are displayed if *rvoShowSpecialCharacters* is included in TCustomRichView.Options 146, and *rvscParagraph* is included in RVVisibleSpecialCharacters 665 variable.

Value	Meaning
rvpmStandard	Pilcrow for paragraph breaks, bended arrow for line breaks
rvpmArrows	Bended arrow for paragraph breaks, arrow down for line breaks

Example:



See also:

- TRVStyle.SpecialCharactersColor 402

1.9.28 rvsXXX constants

There are three groups of **rvs***** constants:

- identifiers of types of RichView 128 items 29;
- indices in the collection of text styles (RVStyle.TextStyles 404) corresponding to the initial entries;
- values used by TRichView internally when saving and loading RVF 59 files (see RVF specification 77).

Identifiers of Item Types

Use the first group of constants to compare with values returned by RichView.GetItemStyle 194 or TRichViewEdit.CurItemStyle 289.

Constant	Value	Item Type	Unit
rvsBreak	-1	break 93 (horizontal line)	RVStyle
rvsPicture	-3	picture 90	RVStyle
rvsHotspot	-4	hotspot 97 (picture from ImageList - hyperlink)	RVStyle
rvsComponent	-5	inserted Delphi control 94	RVStyle
rvsBullet	-6	bullet 96 (picture from ImageList)	RVStyle
rvsHotPicture	-10	hot picture 93 (picture - hyperlink)	RVStyle
rvsListMarker	-11	list marker 99 (bullet or numbering of paragraphs)	RVStyle
rvsTab	-12	tabulator 89	RVStyle
rvsTable	-60	table 111	RVTable
rvsLabel	-200	label 101 (non-text item looking like text)	RVLabelItem
rvsSequence	-202	numbered sequence 102	RVSeqItem
rvsFootnote	-203	footnote 104	RVNote
rvsEndnote	-204	endnote 103	RVNote
rvsSidenote	-206	sidenotes 105	RVSidenote
rvsTextBox	-207	text boxes 107	RVSidenote
rvsNoteReference	-205	reference to the parent note 108	RVNote
rvsPageNumber	-208	"page number" field 109	RVFieldItems

rvsPageCount	-209	"page count" field 110	RVFieldItems

All these constants are negative so you can distinguish them from indices in the collection of text styles.

See also: RichView item types 29.

Indices in The Collection of Text Styles

The second group of constants is almost obsolete, because the status of these text styles is changed from "standard" (in older versions) to "initial". You can completely customize collections of text styles by removing initial items and adding new items.

All these constants are defined in RVStyle unit.

Constant	Value	Suggested Using
rvsNormal	0	Normal text
rvsHeading	1	Heading 1
rvsSubheading	2	Heading 2
rvsKeyword	3	Keywords
rvsJump1	4	Hypertext 1
rvsJump2	5	Hypertext 2

It's still highly recommended to use the 0-th style as the default (normal) text. For example, this style is used as a default style in exported HTML files.

There is a special constant *rvsDefStyle*.

Constant	Value	Meaning
rvsDefStyle	MaxInt	"Use default text style" **(deprecated!)**

This constant can be used as a StyleNo parameter for methods that add text in TRichView.

When this value is specified instead of the index in the collection of text styles, the paragraph's default text style 457 will be used.

1.9.29 RVThumbnailMaker

Unit RVThumbMaker.

Syntax

```
var RVThumbnailMaker: TRVThumbnailMaker 615;
```

(introduced in version 15)

This variable returns an object used by TRichView components to create smooth scaled images. This object has methods and properties allowing to restrict size of scaled images, specify classes of scaled images, etc.

See also:

- smooth image scaling 57

1.9.30 RVVisibleSpecialCharacters

Unit RVStyle.

Syntax

```
type
  TRVSpecialCharacter = (rvscSpace, rvscNBSP, rvscTab, rvscParagraph,
    rvscSoftHyphen, rvscPlaceholders, rvcFloatingLines);
  TRVSpecialCharacters = set of TRVSpecialCharacter;
const
  RVVisibleSpecialCharacters: TRVSpecialCharacters = [rvscSpace..rvcFloatingLines];
```

(introduced in version 10)

Lists characters which will be shown when *rvoShowSpecialCharacters* is included in TCustomRichView.Options [146].

Value	Meaning
rvscSpace	If included, space characters are shown (as middle dots). Hidden spaces at the ends of lines are not shown, see RichViewShowGhostSpaces [656]
rvscNBSP	If included, non breaking space characters are shown (as circles)
rvscTab	If included, tab characters are shown (as arrows)
rvscParagraph	If included, end-of-paragraph and end-of-line marks are shown (as pilcrows and arrows, respectively)
rvscSoftHyphen	If included, soft hyphen characters are shown in Unicode text.
rvscParagraphAttrs	If included together with *rvscParagraph*, additional icons are drawn next to end-of-paragraph marks: ▪ triangle-arrow showing paragraph direction [450] (if defined); ▪ rectangle if any non-default printing option [453] is defined (never displayed for headings 1...9) ▪ heading level (1...9)
rvscPlaceholders	If included, special marks are painted at places of insertion of text box items [107]
rvcFloatingLines	If included, floating lines are displayed for left-and right-aligned items [648], see TRVStyle.FloatingLineColor [390]

See also:

- RVParagraphMarks [662]
- TRVStyle.SpecialCharactersColor [402]

1.10 How To...

How to...

(printing)
- Change page size and orientation for printing RichView [666]
- Draw page numbers, headers and footers [666]

(editing)
- Implement commands like "make bold", "apply font", etc. [667]
- Implement Find and Replace dialogs [667]
- Switch insert/overtype mode [668]
- Move the caret to the beginning or to the end of document in editor [668]
- Make Unicode editor [669]

- Make a plain text editor 669
- Implement smart indents 671

(miscellaneous)

- Use third-party graphic classes with RichView 671
- Combine several documents in one 671
- Remove all formatting 673

1.10.1 change page size and orientation

TRVPrint 479 uses global printer settings of application.

These settings can be modified by standard Delphi/C++Builder components: TPrintDialog, TPrinterSetupDialog (located on Dialogs page of the Component Palette).

You can change them from code using the global Printer object (unit Printers). For example, you can choose printer using Printer.PrinterIndex, or change page orientation using Printer.Orientation.

Changes will be in effect after calling TRVPrint.FormatPages 490 method.

1.10.2 draw page numbers, headers and footers

Complex headers and footers, the same for all pages

The component supports complex headers and footers, they can be multiline, contain images, tables, etc.

They must be assigned to TRVPrint 479 component by the methods SetHeader 493 and SetFooter 493.

See the example:

- Demos\Delphi\Assorted\Printing\Headers\
- Demos\CBuilder\Assorted\Printing\Printing\

There is a limitation of this method: the same header and footer for all pages, with no dynamic fields (like page number).

Drawing headers and footers yourself

If you need different headers for different pages, you need to draw them yourself, using OnPagePrepaint 495 event.

This demo shows how to draw a simple text as a header:

- Demos\Delphi\Assorted\Printing\Printing\
- Demos\CBuilder\Assorted\Printing\Printing\

Since you draw this text yourself, you can draw as many lines as you want.

RichViewActions process this event themselves and draw one plain text line. They understand several codes like page number and pages count.

Drawing complex headers and footers using TRVReportHelper

If you need to draw more complex headers (different for different pages), you can use TRVReportHelper 503 component for this:

http://www.trichview.com/support/files/printheader.zip (for Delphi)

This demo uses TRVReportHelper to draw header in TRVPrint.OnPagePrepaint 495 event.

This demo draws the same header for all pages, but can be easily changed to draw different ones.

It is is shown in this demo:

http://www.trichview.com/support/files/cppheader.zip (for C++Builder).

The next demo uses TRVReportHelper for drawing both header/footer and the main document:

http://www.trichview.com/support/files/a4.zip (for Delphi)

It draws the result on the metafile, but can be easily changed to draw on the printer canvas.

The most complex case:

- headers are complex documents;
- headers have different heights on different pages.

http://www.trichview.com/support/files/printheaderex.zip (for Delphi)

Online updates

New information on this topic will be posted here: ***http://www.trichview.com/forums/viewtopic.php?t=327***

1.10.3 implement commands like "make bold", "apply font", etc.

The key method is ApplyStyleConversion [299].

Example:

- Demos\Delphi\Editors\Editor 2\
- Demos\CBuilder\Editors\Editor 2\

Alternatively, you can use ***RichViewActions***.

1.10.4 implement Find and Replace dialogs

Find

The key method is TRichView.SearchText [222] or TRichViewEdit.SearchText [336].

Search can be case sensitive or not, up or down. "Whole word" option is also supported.

You can convert options from Find dialog to SrchOptions parameter of these methods using GetRVSearchOptions [616] or GetRVESearchOptions [616].

Replace

Use TRichViewEdit.SearchText [336] to select the old string.

Then use InsertText [327] to replace it with the new one. This method removes the selected part of document before performing insertion. If you search in backward direction, set CaretBefore parameter to *True*.

Examples

Demos:

Find dialog is implemented in the main editor demo:

- Demos\Delphi\Editors\Editor 1\
- Demos\CBuilder\Editors\Editor 1\

Both Find and Replace dialogs are implemented in the demo:

- Demos\Delphi\Assorted\Search and Replace\
- Demos\CBuilder\Assorted\Search and Replace\

Alternatively, you can use ***RichViewActions***.

1.10.5 switch insert/overtype mode

The overwrite mode is not supported directly, but you can implement it using events.

The code is below.

```
var IgnoreNextChar: Boolean = False;

procedure TForm1.RichViewEdit1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  IgnoreNextChar := RichViewEdit1.SelectionExists[224];
end;

procedure TForm1.RichViewEdit1KeyPress(Sender: TObject; var Key: Char);
var rve: TCustomRichViewEdit[282];
    ItemNo, Offs: Integer;
begin
  if IgnoreNextChar then begin
    IgnoreNextChar := False;
    exit;
  end;
  IgnoreNextChar := False;
  if not ((Key=#9) or (Key>=' ')) then
    exit;
  rve := RichViewEdit1.TopLevelEditor[295];
  if rve.SelectionExists[224] then
    exit;
  ItemNo := rve.CurItemNo[288];
  Offs  := rve.OffsetInCurItem[294];
  if (Offs>=rve.GetOffsAfterItem[200](ItemNo)) then begin
    if (ItemNo+1<rve.ItemCount[144]) and
       not rve.IsFromNewLine[207](ItemNo+1) then begin
      inc(ItemNo);
      Offs := rve.GetOffsBeforeItem[201](ItemNo);
      end
    else
      exit;
  end;
  rve.SetSelectionBounds[235](ItemNo, Offs, ItemNo, Offs+1);
  rve.Invalidate;
end;
```

More details and demo: ***http://www.trichview.com/forums/viewtopic.php?t=171***

1.10.6 move the caret to the beginning or to the end of document in editor

The caret is always at the end of selection.

Selection can be changed using method TRichViewEdit.SetSelectionBounds[235].

Moving to the beginning:

```
var
  ItemNo, Offs: Integer;
...
ItemNo := 0;
Offs   := MyRichViewEdit.GetOffsBeforeItem[201](ItemNo);
MyRichViewEdit.SetSelectionBounds(ItemNo,Offs,ItemNo,Offs);
```

Moving to the end:

```
var
  ItemNo, Offs: Integer;
...
ItemNo := MyRichViewEdit.ItemCount[144]-1;
Offs   := MyRichViewEdit.GetOffsAfterItem[200](ItemNo);
MyRichViewEdit.SetSelectionBounds(ItemNo,Offs,ItemNo,Offs);
```

See also:

- TRichViewEdit.CurItemNo[288] and OffsetInCurItem[294];
- Example how to move caret to the beginning of paragraph[371].

1.10.7 make Unicode editor

See in the topic Unicode in TRichView[64]..

1.10.8 make a plain text editor

How to make a plain text editor based on TRichViewEdit[282]

- Exclude all but text from AcceptDragDropFormat[287] property.
- Process OnPaste[365] event (see below).
- Disable all UI commands changing text and paragraph attributes.

```
procedure TMyForm.MyRichViewEditPaste(Sender: TCustomRichViewEdit;
  var DoDefault: Boolean);
var s: String;
begin
  if Clipboard.HasFormat(CF_TEXT) then begin
    s := Clipboard.AsText;
    Sender.InsertText[327](s, False);
  end;
  DoDefault := False;
end;
```

How to make ONE LINE plain text editor

- Exclude all but text from AcceptDragDropFormat[287] property.
- Process OnPaste[365] event (see below).
- Process OnOleDrop[363] event (see below).
- Disable all UI commands changing text and paragraph attributes or inserting multiline text.
- Set WordWrap[159] property = *False*.
- Include *rvoDoNotWantReturns* in EditorOptions[291].
- Set VSmallStep[159] property = 1, before calling Format[182].
- If you do not want tabulators as well, set RVStyle.SpacesInTab[402] = 4.

```
uses Clipbrd, ActiveX;
// Returning one line string made from s.
// This function replaces all line breaks with ' | '.
// You can change this function, for example to return the first line
function MakeOneLineString(const s: String): String;
var p: Integer;
begin
  Result := AdjustLineBreaks(s);
  while True do begin
    p := Pos(#13#10, Result);
    if p=0 then
      break;
```

```
    Delete(Result, p, 2);
    Insert(' | ', Result, p);
 end;
end;

procedure TMyForm.MyRichViewEditOleDrop(Sender: TCustomRichView[128];
  const DataObject: IDataObject; Shift: TShiftState; X, Y: Integer;
  PossibleDropEffects: TRVOleDropEffects[636]; var DropEffect: TRVOleDropEffect[636];
  var DoDefault: Boolean);
var FmtEtc: TFormatEtc;
  StgMedium: TStgMedium;
  ptr: Pointer;
  s: String;
  p, ByteCount, CharCount: Integer;
begin
  DoDefault := False;

  FillChar(StgMedium, sizeof(StgMedium), 0);
  FillChar(FmtEtc, sizeof(FmtEtc), 0);
  FmtEtc.cfFormat := CF_TEXT; // CF_UNICODETEXT for Delphi 2009+
  FmtEtc.dwAspect := DVASPECT_CONTENT;
  FmtEtc.lindex := -1;
  FmtEtc.tymed := TYMED_HGLOBAL;
  if DataObject.GetData(FmtEtc, StgMedium)<>S_OK then
    exit;
  if StgMedium.tymed=TYMED_HGLOBAL then begin
    ptr := GlobalLock(StgMedium.HGlobal);
    try
      ByteCount := GlobalSize(StgMedium.HGlobal);
      CharCount := ByteCount;
      // CharCount := CharCount div 2; // uncomment for Delphi 2009+
      SetLength(s, CharCount);
      Move(ptr^, PChar(s)^, ByteCount);
    finally
      GlobalUnlock(StgMedium.HGlobal);
    end;
    p := Pos(#0, s);
    if p>0 then
      s := Copy(s, 1, p-1);
    MyRichViewEdit.InsertText[327](MakeOneLineString(s), False);
  end;
  ReleaseStgMedium(StgMedium);
end;

procedure TMyForm.MyRichViewEditPaste(Sender: TCustomRichViewEdit[282];
  var DoDefault: Boolean);
var s: String;
begin
 if Clipboard.HasFormat(CF_TEXT) then begin
    s := Clipboard.AsText;
    Sender.InsertText[327](MakeOneLineString(s), False);
  end;
  DoDefault := False;
end;
```

1.10.9 implement smart indenting

You can find the examples in the support forum: ***http://www.trichview.com/forums/viewtopic.php?t=2044***

There are two examples available.

Indenting in programming style

When the user presses **Enter**, the code adds space characters at the beginning of the new paragraph (as many spaces as at the beginning of the current paragraph).

Indenting in Microsoft Word style

The code changes FirstIndent[451] and LeftIndent[451] of paragraph, if **Tab** / **Backspace** is pressed when the caret is at the beginning of paragraph. First, it changes FirstIndent[451]. Next, if FirstIndent[451] is already changed, it changes LeftIndent[451]. The code changes indents only if the caret is at the beginning of non-empty paragraph.

1.10.10 combine several TRichView documents in one

See in the topic TCustomRichView.InsertRVFFromStream[206].

1.10.11 use third-party graphic classes with RichView

Using third party graphic classes

You can use graphic classes created by third party developers.

These classes can be used just like standard classes (TBitmap, TIcon, TMetafile, TJpegImage).

You can add picture in a document using AddPictureEx[169] or InsertPicture[322].

Example:

```
gif: TGifImage;
...
gif := TGifImage.Create;
gif.LoadFromFile('demo.gif');
editor.InsertPicture('Demo', gif, rvvaBaseLine);
```

For example, you can use free graphic classes:

- Anders Melander's TGifImage ***www.torry.net/vcl/graphics/gif/gifimage.exe*** (update for Delphi 6 and newer: ***www.trichview.com/resources/thirdparty/gifimage.zip***);
- TJvGifImage from JEDI's JVCL ***http://jvcl.sourceforge.net***;
- Gustavo Huffenbacher Daud's TPngObject ***www.trichview.com/resources/thirdparty/pngimage.zip***.

Starting from Delphi 2007, an advanced version of Anders Melander's TGifImage is included in VCL, so TGifImage becomes a standard graphic class.

Starting from Delphi 2009, TPngImage is included in VCL.

Gif animation

To support animation[56], assign TCustomRichView.AnimationMode[135] = *rvaniOnFormat*.

The following units must be included in your projects:

- RVGifAnimate for Anders Melander's TGifImage;
- RVJvGifAnimate for TJvGifImage from JEDI's JVCL;
- RVGifAnimate2007 for standard Delphi TGifImage (in Delphi 2007 or newer).

See also: animation in TRichView[56]

Controlling automatic insertion of graphics when importing RTF

Sometimes images are inserted automatically (for example, when importing images referred from RTF).

Question: What graphic classes does TRichView use for loading non-standard graphic formats (such as GIF or PNG)?

Answer: RichView uses the graphic class which is registered for the given file extension.

If a third party creator has not associated his graphic class with the extension, you can do it yourself with TPicture.RegisterFileFormat:

```
TPicture.RegisterFileFormat('gif','Gif Image',TGifImage);
```

This registration is especially necessary if you define this graphic format as HTML graphic format [672].

Workaround for Delphi bug in constructors of graphic classes (Delphi 5 or older)

If you use Delphi 5/C++Builder 5 or older, use RVGraphicHandler [662].CreateGraphic (defined in RVFuncs unit) instead of direct call of graphic class constructor.

It is not necessary if you use Delphi 6/C++Builder 6 or newer.

Exporting graphics in HTML

By default, all images are converted to JPEGs. This is obviously undesirable for formats like GIF and PNG.

You can specify which graphic classes must not be converted.

Use RVGraphicHandler [662].RegisterHTMLGraphicFormat procedure, for example:

```
uses RVGrHandler;
...
// call this before the first html export
RVGraphicHandler.RegisterHTMLGraphicFormat(TMyGifImage);
```

This registration affects only images of the given format. Other formats will be converted to JPEGs.

TRichView registers the following classes as HTML graphic formats automatically:

- standard TPngImage (for Delphi 2009+);
- standard TGifImage (for Delphi 2007+), if you include RVGifAnimate2007 unit in your project;
- Anders Melander's TGifImage, if you include RVGifAnimate unit in your project;
- TJvGifImage from JEDI's JVCL, if you include RVJvGifAnimate unit in your project.

Specifying PNG class

Specify graphic class for PNG images using Use RVGraphicHandler [662].RegisterPngGraphic, for example:

```
uses RVFuncs;
...
RVGraphicHandler.RegisterPngGraphic(TPngObject).
```

It allows loading PNG images from RTF files and saving PNG images to RTF and DocX files without converting to bitmaps or metafiles.

In Delphi 2009+, TRichView automatically registers the standard TPngImage as a PNG class.

Specifying JPEG class

Specify graphic class for Jpeg images using Use RVGraphicHandler [662].RegisterJpegGraphic, for example:

```
uses RVFuncs;
...
RVGraphicHandler.RegisterJpegGraphic(TMyJpegImage).
```

If you do not call this method, TRichView uses TJPEGImage.

This class is used when loading JPEG images embedded in RTF.

In Delphi 2009+, TRichView automatically registers the standard TPngImage as a PNG class.

Using DevExpress graphic classes

If you uncomment RVUSEDXPNGIMAGE in RV_Defs.inc, TRichView supports DevExpress classes:

- TcxPngImage for Png
- TdxSmartImage for Bitmap, Jpeg, Tiff, Gif images.

TcxPngImage becomes a default Png class for TRichView, but TdxSmartImage does not become a default Jpeg class, unless you call RVGraphicHandler.RegisterJpegGraphic explicitly.

1.10.12 create a chat window

You can find the complete demo of a chat window on the support forum: ***http://www.trichview.com/forums/viewtopic.php?t=63***

This demo shows how to implement a chat window with the following features. The user types a plain text message in TEdit. This message can contain special commands, they are processed before displaying in TRichView window.

This demo supports:

- text commands;
- emoticons;
- URL detection.

Examples of text commands:

- **\b** to make text bold (toggle);
- **\cRRGGBB** to change text color.

Emoticons are easily customizable (their codes and graphics are stored in arrays). This demo uses bitmaps, but you can change them to Gif images (see how to use third-party graphic classes 671), including animations 56. For additional ideas about emoticons, see the demo in **Demos\Delphi\Assorted\Graphics\Emoticons**.

URL detection use RVIsURL and RVIsEmail 619 functions. You can use RVIsCustomURL 662 to define your own types of links.

1.10.13 remove text formatting

The code below removes text formatting from the whole document.

It converts indices of all text styles 404 and paragraph styles 398 to 0s. Optionally, it can keep hyperlinks (they are converted to the hypertext style with the lowest index).

Optionally, it can remove list markers 99 as well.

This code changes text items 88, tabulators 89 and list markers. All other item types remains unchanged.

Note 1: if the document contains both Unicode 64 and non-Unicode text, or non-Unicode text of different Charsets, some text information may be lost.

Note 2: this is not an editing operation. When called for TRichViewEdit, undo buffer must be cleared.

```
// this code uses some undocumented methods
procedure RemoveFormatting(RVData: TCustomRVData608; RemoveMarkers, KeepLinks: Boolean);
var i, HypertextStyleNo: Integer;
    UnicodeTo, UnicodeHTo: Boolean;
  {.....................................................}
  procedure DoRemoveFormatting(RVData: TCustomRVData608);
  var i, r, c, StyleNo, StyleNoTo: Integer;
      PB, UnicodeFrom, ThisUnicodeTo: Boolean;
```

```
    table: TRVTableItemInfo;
begin
  for i := RVData.ItemCount[144]-1 downto 0 do begin
    RVData.GetItem[190](i).ParaNo := 0;
    StyleNo := RVData.GetItemStyle[194](i);
    case StyleNo of
      rvsTable[663]:
        begin
          table := TRVTableItemInfo[528](RVData.GetItem[190](i));
          for r := 0 to table.RowCount[543]-1 do
            for c := 0 to table.ColCount[538]-1 do
              if table.Cells[537][r,c]<>nil then
                DoRemoveFormatting(table.Cells[537][r,c].GetRVData[575]);
        end;
      rvsListMarker[663]:
        if RemoveMarkers then begin
          PB := RVData.PageBreaksBeforeItems[149][i];
          RVData.DeleteItems[179](i, 1);
          if i<RVData.ItemCount[144] then begin
            RVData.GetItem[190](i).SameAsPrev := False;
            RVData.GetItem[190](i).PageBreakBefore := PB;
          end;
        end;
      rvsTab[663]:
        if RVData.GetRVStyle.TextStyles[404][
        TRVTabItemInfo[526](RVData.GetItem[190](i)).TextStyleNo].Jump[445] and KeepLinks then
          TRVTabItemInfo(RVData.GetItem[190](i)).TextStyleNo := HypertextStyleNo
        else begin
          TRVTabItemInfo(RVData.GetItem[190](i)).TextStyleNo := 0;
          RVData.SetItemTag[231](i, '');
        end;
      1..MaxInt:
        begin
          if KeepLinks and RVData.GetRVStyle.TextStyles[404][StyleNo].Jump[445] then begin
            ThisUnicodeTo := UnicodeHTo;
            StyleNoTo := HypertextStyleNo;
            end
          else begin
            ThisUnicodeTo := UnicodeTo;
            StyleNoTo := 0;
            RVData.SetItemTag[231](i, '');
          end;
          UnicodeFrom := RVData.GetRVStyle.TextStyles[404][StyleNo].Unicode[446];
          RVData.GetItem[190](i).StyleNo := StyleNoTo;
          if UnicodeFrom and not ThisUnicodeTo then begin
            RVData.GetItem[190](i).ItemOptions := RVData.GetItem(i).ItemOptions-[rvioUnicode
            RVData.SetItemTextR[232](i,
              RVU_UnicodeToAnsi(RVData.GetStyleCodePage(0), RVData.GetItemTextR[195](i)));
            end
          else if not UnicodeFrom and ThisUnicodeTo then begin
            RVData.GetItem[190](i).ItemOptions := RVData.GetItem(i).ItemOptions+[rvioUnicode
            RVData.SetItemTextR[232](i,
              RVU_AnsiToUnicode(RVData.GetStyleCodePage(0), RVData.GetItemTextR[195](i)));
          end;
        end;
    end;
  end;
```

```
  end;
  {.....................................................}
begin
  HypertextStyleNo := 0;
  if KeepLinks then
    for i := 0 to RVData.GetRVStyle.TextStyles[404].Count-1 do
      if RVData.GetRVStyle.TextStyles[404][i].Jump[445] then begin
        HypertextStyleNo := i;
        break;
      end;
  UnicodeTo  := RVData.GetRVStyle.TextStyles[404][0].Unicode[446];
  UnicodeHTo := RVData.GetRVStyle.TextStyles[404][HypertextStyleNo].Unicode[446];
  DoRemoveFormatting(RVData);
end;
```

How to use:

```
  RemoveFormatting(RichViewEdit1.RVData[151], True, True);
  NormalizeRichView(RichViewEdit1.RVData[151]); // from RVNormalize.pas, from RichViewActio
  RichViewEdit1.DeleteUnusedStyles[181](True, True, True);
  RichViewEdit1.ClearUndo[305];
  RichViewEdit1.Format[182];
```

More information: ***http://www.trichview.com/forums/viewtopic.php?t=2392***

1.11 Version History

Version History and Compatibility Issues

- New after version 16 675
- New in version 16 676
- New in version 15 678
- New in version 14 679
- New in version 13 682
- New in version 12 684
- New in version 11 685
- New in version 10 687
- New in version 1.9 691
- New in version 1.8 693
- New in version 1.7 695
- New in version 1.6 698
- New in version 1.5 700
- New in version 1.4 702
- New in version 1.3 703
- New in version 1.2 704
- Compatibility with older (freeware) version 705

1.11.1 New after version 16

Compatibility issues

New parameter in TRVPageCountItemInfo[607].CreateEx[607]

Tables[528]

New feature: colors of groups or rows and columns (alternate colors). New properties:

- HeadingRowColor[546], LastRowColor, FirstColumnColor, LastColumnColor, EvenColumnsColor,

OddRowsColor, OddColumnsColor, EvenRowsColor;
- RowBandSize, ColBandSize[538].

Printing

"Widow and orphan control" is implemented, see *rvpaoWidowOrphanControl*[453] option of paragraphs.

TRVPageCountItemInfo[607] now has NumberType[605] property, like TRVPageNumberItemInfo[605],

1.11.2 New in version 16

Compatibility issues

- Since this version, text measuring and drawing is performed using ***Uniscribe***; so text size may be different (for example, Uniscribe always uses text kerning); usually, text formatting is slower with Uniscribe (if you do not need support of bidirected text); you can switch back to Windows API, see "Uniscribe" below
- Uniscribe text output (and sometimes Windows text output) is not compatible with metafiles; see "Uniscribe" below;
- ParaStyle.Alignment[449] = *rvaJustify* is implemented by modifying space character widths when drawing text, while older versions of the components drew each word separately. The new method complicates custom drawing of text[415]; you can return the old justification method by activating RVOLDJUSTIFY in RV_Defs.inc; however, this directive disables ParaStyle.Alignment[449] = *rvaDistribute* (it works like *rvaJustify*)
- TRVStyle.OnDrawStyleText[415] has new parameters.
- RichViewSafeFormatting constant is removed (replaced with RichViewStringFormatMethod[656])
- applications compiled with older versions of TRichView cannot read RVF files containing tables having changed values of Opacity (Opacity<>100000).
- starting from this version, all projects for C++Builder 6 that use TRichView must include $(BCB)\Lib\Psdk \msimg32.lib. Otherwise, linking ends with an error about unresolved external "AlphaBlend". No changes are required for Delphi and newer versions of C++Builder
- the algorithm for calculation of width of table columns is improved and may produce different results.

Uniscribe

Since this version, the components use ***Uniscribe*** for text drawing and measuring.

You can switch back to Windows API using TRVStyle.TextEngine[403] property.

By default, Uniscribe text output is not compatible with metafiles. You can turn on metafile compatibility for TRVReportHelper[503] and TRVPrint[479] components using MetafileCompatibility[515] property.

New paragraph properties

New value for Alignment[449] – *rvaDistribute*. It aligns paragraph content to the both left and right sides by adding space between all characters.

New property LastLineAlignment[449] defines alignment for the last line of justified and distributed paragraphs.

Improved attributes in bi-directed text

The following features were previously disabled for bi-directed text; they work now:
- TextStyle.CharSpacing[435]
- ParaStyle.Alignment[449] = *rvaJustify* (and *rvaDistribute*)
- TRVPrint.PreviewCorrection[516]
- custom drawing of selection (although, custom drawing is more complicated now)

All these functions are implemented both for Windows and Uniscribe API.

Vertical text

Special options[571] are implemented for vertical text in table cells.

In addition to rotation[573] by 90°, the component can automatically apply vertical versions of fonts (useful for Chinese, Japanese, Korean) and reverse line order (useful for traditional Mongolian)

Formatting

The components need less memory for documents containing justified text.

Improvements

- documents containing long strings in text items are formatted much faster than before; improved fast text formatting method now can be used not only in TRichViewEdit, but in TRVPrint and in ScaleRichView;
- justified text is formatted much faster than before.

Table sizing algorithm is improved, it processes wide spanned cells much better.

Styles

New RVFOption[152]: *rvfoSaveStyleTemplatesOnlyNames*. It allows creating multiple RVF documents using the same collection of StyleTemplates[402].

Transparency

Background of paragraphs, tables, table cells, sidenotes and text boxes can be semitransparent: a new Opacity property is added to these objects.

New property TRVReportHelper.AllowTransparentDrawing[504] allows using methods for semitransparent drawing. It is useful for drawing over other images.

See the topic about semitransparent objects[57].

Support of high DPI screen modes

Size of touch screen selection handles[398] is changed proportionally to the screen resolution. Sizes of list markers are stored as Size (instead of Height) property in DFM and RVF files, so they do not depend on screen resolution any more.

Installation and directory structure

Starting from this update, TRichView is installed automatically in Delphi and C++Builder IDE.

The new installer installs the components in Delphi and C++Builder, both for 32-bit and 64-bit platforms (if available). The installer adds all necessary paths to RAD Studio library.

Source code is moved to "Source" folder, inc-file is moved to "Source\Include" folder.

For RAD Studio XE8+, this help file is integrated in IDE (both F1 help, and menu "Help | Third-Party Help")

Other changes

TRichViewEdit.RemoveCurrentPageBreak[334] deletes a page break from the current paragraph (not from the current item as before)

New RichViewSaveHyperlinksInDocXAsFields[656] global variable.

Ctrl + **Delete** deletes a word to the right of the caret.

1.11.3 New in version 15

Compatibility issues

- The following graphic classes are registered automatically (Do not call RegisterClass or RegisterClassAlias for these classes):
  - (for Delphi 2009+) TPngImage. Also, TPngImage is automatically registered as the default PNG format and as HTML graphic format (see TRVGraphicHandler[613])
  - (for Delphi 2010+) TWicImage. Also, thumbnails are activated by default for TWicImage.
  - (for Delphi 2007+) standard TGifImage – if you include RVGifAnimate2007 unit in your project;
  - Anders Melander's TGifImage – if you include RVGifAnimate unit in your project;
  - TJvGifImage from JEDI's JVCL – if you include RVJvGifAnimate unit in your project.
- A parameter is added to TRVPrint.GetHeaderRect[490], GetFooterRect[490].
- New values are added to TRVUndoType[647].

New types of items

- sidenotes[105] and text box items[107] are printed in floating boxes.
- "page number" fields[109]
- "page count" fields[110]

DocX export

TRichView can save Microsoft Word 2007+ files (*.docx).

Minimal required versions for this feature: Delphi: 4; C++Builder: 2006.

New methods of TCustomRichView[128]: SaveDocX[215], SaveDocXToStream[216].

New OnSaveDocXExtra[257] event.

New options for TCustomRVFontInfo[432].Options[438]: *rvteoDocXCode* and *rvteoDocXInRunCode.*

New properties of TRVOfficeConverter[496]: ExcludeDocXImportConverter, ExcludeDocXExportConverter[498].

Touch screen support

Multi-touch screen operations are supported (for Delphi 2010+). The standard touch-related properties and events (Touch and OnGesture) are published.

All visual controls support a panning gesture. TRVPrintPreview[384] supports a two finger zooming as well.

Selection handles are implemented in TRichView[128] and TRichViewEdit[282]. They are shown for the selection if it was created by a double (word selection) or a triple (paragraph selection) tap. Alternatively, you can show and hide them using SelectionHandlesVisible[156] property.

New property: TRVStyle.SelectionHandleKind[398].

Printing and drawing

New properties defining sizes in TRVPrint[479]: Margins[486], HeaderY[485], FooterY[485], Units[487]. The new properties replace LeftMarginMM, TopMarginMM, RightMarginMM, BottomMarginMM, HeaderYMM, FooterYMM properties. The old properties still work. The main advantage of the new properties is a higher precision. Even if Units is millimeters, you can define properties more accurately, because sizes are floating point values.

Now you can define different headers and footers for the first page, and for odd/even pages. TRVPrint.SetHeader[493] and SetFooter[493] have additional optional parameter, specifying header/footer type. New properties: TitlePage[487], FacingPages[484].

New property: PageNoFromNumber[516] specifies the starting value for "page number" fields[109]. In TRVReportHelper[503], page numbers can be requested in OnGetPageNumber[508] event.

New TRVReportHelper[503].NoMetafiles property allows drawing metafiles as bitmaps.

Editing

New methods allowing to change properties of TRichViewEdit as an editing (undoable) operation: SetIntPropertyEd, SetStrPropertyEd, SetFloatPropertyEd[337], SetBackgroundImageEd[338]. They allow changing margins, DocParameters, background properties.

New methods allowing to change properties of items in TRichViewEdit as an editing (undoable) operation:

- SetCurrentItemExtraIntPropertyEx[345], SetItemExtraIntPropertyExEd[345];
- SetCurrentItemExtraStrPropertyEx[346], SetItemExtraStrPropertyExEd[346].

They support more properties than the old methods.

Text attributes

New property SizeDouble[441] allows defining font size with higher precision.

Visual changes

- New TRVStyle[385].FontQuality[391] property.
- Smooth image scaling[57].

Other changes

Packages for Delphi and C++Builder XE4, XE5, XE6, XE7 are added.

Footnotes[104]/endnotes[103] displaying a character (instead of a number) are supported.

All work with graphic classes is encapsulated in TRVGraphicHandler[613] class, available via RVGraphicHandler[662] variable.

RichViewJustifyBeforeLineBreak[654] variable allow to turn on justifying lines before line breaks.

Faster processing of hidden text.

New properties:

- TRVScroller[508].NoHScroll[511];
- TCustomRichViewEdit[282].ForceFieldHighlight[293].

New option for TCustomRichView[128].Options[146]: *rvoPercentEncodedURL*.

1.11.4 New in version 14

Compatibility issues

- Item tags[34] are strings, they are not integers any more. The default tag value is changed from 0 to ''. If you used tags in an integer mode, use StrToInt and IntToStr when calling TRichView methods returning and accepting tags. Not only methods having Tag parameters, but some events are affected (OnReadHyperlink[248], OnRVControlNeeded[251]). In existing projects, change the types of Tag parameters of these events handlers from Integer to TRVTag manually. RVOLDTAGS can be defined in RV_Defs.inc to make tags integers (not recommended).
- Parameters of OnRVFPictureNeeded[252] were changed. If you used this event, change events handlers manually (or delete the old event handler and create a new one).
- For rotated cells, coordinates stored in DrawItems (a low-level undocumented property) do not define the position of items on the screen directly.
- TRichView.LoadText[212], LoadTextFromStream[213], SaveText[221], SaveTextToStream[221], TRichViewEdit.InsertTextFromFile[328] have a new parameter. This parameter is optional, but in old versions of C++Builder, it must be added explicitly in the code (use CP_ACP for compatibility).

- A text search behavior can be modified with the new "smart start" option (see TRichView.SearchText[222] and TRichViewEdit.SearchText[336]). GetRVSearchOptions[616] and GetRVESearchOptions[616] include this option by default, so it may affect existing projects.
- Changes in Unicode line wrapping procedure: '.' , '!', ',', ';' characters are treated (almost) as alphabetic characters now (because they are often used in phrases like '.Net' or in URLs).
- Changes in DB components: If *rvfoLoadBack* is in RVFOptions[152], TDBRichView[373] clears BackgroundBitmap[136] before loading data from its field. If *rvfoLoadBack* is in RVFOptions[152] and FieldFormat[380]=*rvdbRVF*, TDBRichViewEdit[377] clears BackgroundBitmap[136] before loading data from its field.
- PNG images are saved to RTF as PNG images only if *rvrtfSavePngAsPng* is included in RTFOptions[150]. It is included by default. However, for existing projects, you may need to set it manually.
- The declaration of TRVLongOperation[248] type is changed.
- The following global variables were removed from RVTable.pas: RichViewTableGridStyle, RichViewTableGridStyle2, RichViewTableGridColor. They were replaced with properties of TRVStyle[392].
- A new option *rvoShowGridLines* in TRichView.Options[146] is set by default, but for existing projects you may need to set it manually, otherwise grid lines of tables will not be shown by default.

Style templates

StyleTemplates[402] represent "real" text and paragraph styles. When style templates are used, an appearance of text and paragraphs are still defined in TextStyles[404] and ParaStyles[398]. Items a document refer to items in TextStyles and ParaStyles, while items in TextStyles and ParaStyles refer to items in StyleTemplates. Style templates allows modifying TextStyles and ParaStyles.

By default, style templates are not used. To activate using of style templates, assign TRichView.UseStyleTemplates[158]=*True*.

TRichView.StyleTemplateInsertMode[156] specifies how style templates affect insertion of RVF and RTF data.

When the collection of style templates has been modified, OnStyleTemplatesChange[266] occurs.

In TRichViewEdit, you can apply the specified style template to the selection using the methods: ApplyStyleTemplate[299], ApplyTextStyleTemplate[301], ApplyParaStyleTemplate[298]. Style templates can be edited using ChangeStyleTemplates[304] method.

The function RVGetNoteTextStyleNo[619] has an additional optional parameter: a name of a style template for using for a note character.

TRVStyle.MainRVStyle[397] property allows using a single collection of style templates for several documents.

See the overview topic about style templates[40].

Rotation

Table cells can be rotated[573] by 90°, 180°, or 270°.

A new method returning item coordinates: GetItemCoordsEx[192]; it takes cell rotation into account.

String tags

Since version 13.2, item tags[34] are strings (Unicode for Delphi 2009 or newer, ANSI for older versions of Delphi). No more typecasting from integer to PChar!

New property: cell tags[574].

64-bit

The both 32-bit and 64-bit compilers are supported in RAD Studio XE2+.

Note: TRVOfficeConverter[496] can be compiled in 64-bit application, but lists of converters will be empty (because

converters are 32-bit DLLs and cannot be used in 64-bit applications).

Skins (Delphi XE2+ visual UI styles)

The components support visual UI styles introduced in RAD Studio XE2:

- all visual components display scrollbars and borders according to styles;
- TRVPrintPreview[384] adjusts its colors (background, page border, shadow) according to the current style.

Text files

A new optional parameter for TRichView.LoadText[212], LoadTextFromStream[213], SaveText[221], SaveTextToStream[221], TRichViewEdit.InsertTextFromFile[328]: CodePage. It allows to specify a code page of the file (this parameter is used only for Unicode text in a document; for ANSI text, it is ignored).

Drag and drop: when accepting text files, TRichViewEdit detects their encoding (UTF16, UTF8 or ANSI).

When saving Unicode text, characters of "Symbol" and "Wingdings" fonts are converted to the most similar Unicode characters.

RVF

VAlign property of *bullets*[96] and *hotspots*[97] is saved to RVF.

The parameters of TRichView.OnRVFPictureNeeded[252] were changed. Now, this event is called not only for pictures[90] and hot-pictures[93], but also for tables[98] and table cells to read background images. This change solved the following problems:

- ability to request table and cell background images;
- access to additional item properties.

RTF

New option in TRichView.RTFOptions[150]: *rvrtfSavePngAsPng*. If included (default), PNG images (of the class type registered with RV_RegisterPngGraphic) are saved as PNG in RTF (as it was before this version). You can exclude this option to save them as a bitmap or a metafile.

Default processing if OnReadHyperlink[248]/OnWriteHyperlink[267] is not assigned.

HTML saving

TRichView.SaveHTMLEx[217] can save stretched background images (previously, they were saved as centered).

TRichView.SaveHTMLEx[217] saves list markers better (both in normal and rvsoMarkersAsText[639] modes).

New "hidden" parameters in TRichView.OnSaveImage2[260] event.

Default processing if OnWriteHyperlink[267] is not assigned.

SaveHTMLEx[217] saves characters of "Symbol" and "Wingdings" fonts as the most similar Unicode characters (it was implemented for "Symbol" before, now it is improved).

Visual changes

- Better displaying of text background, new TRVStyle.TextBackgroundKind[402] property.
- All adjacent hypertext items[35] having the same non-empty tags[34] are highlighted below the mouse pointer; so visually they represent a single hyperlink.
- New properties[627] for non-text items: background color, border width and color, spacing around the border.
- Footnote[104] and endnote[103] characters are kept on the same line as the previous item.
- Better calculation of superscript position.

Clipboard

New: TRichViewEdit can paste URL. A new method is available: PasteURL[332].

New property: AcceptPasteFormats[287] allows to restrict a list of formats that the editor can paste.

New property: DefaultPictureVAlign[291] defines an alignment for pasted and dropped images.

DB components

If *rvfoLoadBack* is in RVFOptions[152], TDBRichView[373] clears BackgroundBitmap[136] before loading data from its field.

If *rvfoLoadBack* is in RVFOptions[152] and FieldFormat[380]=*rvdbRVF*, TDBRichViewEdit[377] clears BackgroundBitmap[136] before loading data from its field.

Printing

Tables

New property for a table row: KeepTogether[577]. New table methods: SetRowPageBreakBefore, SetRowKeepTogether[562].

New option for table.PrintOptions[542]: *rvtoContinue*.

Other

New property: TCustomRVPrint.IgnorePageBreaks[515].

Tables

New properties of TRVStyle defining the appearance of grid lines: GridColor, GridStyle, GridReadOnlyColor, GridReadOnlyStyle[392]. They replace global variables used in older versions. New option *rvoShowGridLines* in TRichView.Options[146].

Other changes

Better processing of Up and Down keys when moving in and out of table cells.

New event of TRichViewEdit[282]: OnCheckStickingItems[357] event allows to stick items together.

New typed constant: RichViewDoNotMergeNumbering[654].

"Smart start" search option for TRichView.SearchText[222] and TRichViewEdit.SearchText[336].

TRichView.OnProgress[248] now occurs not only when reading an RTF, but also when saving an RTF, reading and writing an RVF, writing an HTML, reading and writing a text, calling TRVOfficeConverter.ImportRV[502] and ExportRV[500].

Better table layout algorithm.

New methods for TRVStyle: FindTextStyle, FindParaStyle[407], TextStyles.FindSuchStyleEx[422], ParaStyles.FindSuchStyleEx[425].

1.11.5 New in version 13

Compatibility issues

1. This version introduces a new line wrapping algorithm, so layouts of existing documents may become different. The old line wrapping mode can be returned using TRVStyle.LineWrapMode[394] property (*rvWrapSimple* value). But even with this value, a line breaking can be different for Unicode[64] text in paragraphs with Alignment[449]=*rvaJustify*.
   RichViewWrapAnywhere (global constant) was removed, superseded by TRVStyle.LineWrapMode[394]

(*rvWrapAnywhere* value).

2. New parameter in ApplyStyleConversion[299] and ApplyParaStyleConversion[298]. It is optional in Delphi, but for existing C++Builder projects, *True* must be added as the last parameter.
3. This version can write twips instead of pixels in RVF files. Applications compiled with older versions of RichView interprets all sizes an pixels, so if they load such RVF files, all lengths will be too large.
4. This version defines widths of breaks as TRVStyleLength[644] instead of bytes. All applications that use GetBreakInfo[183] and GetCurrentBreakInfo[308] methods must correct the type of variable used read a line width.
5. Paragraph borders are drawn slightly different.

New line wrapping algorithm

New line wrapping algorithm takes connections between text items into account.

New property TRVStyle.LineWrapMode[394] allows changing a line wrapping mode.

Hidden items

Some items can be hidden[42].

Measuring in twips instead of pixels

Previously, all sizes in documents were defined in pixels. Since this version, you can define measuring units in TRVStyle[385].Units[404] property. Instead of pixels, you can use twips[74].

Advantages:

- you can define small widths with better precision than 1 screen pixel;
- you can create user interface allowing entering sizes in different units (fractional inches, cm) and convert them to/from twips without losing precision;
- values in twips are independent from the screen resolution, even if RichViewPixelsPerInch[655]=0.

Shared item content

A new item property is added: rvepShared[627]. It allows for several picture items[90] to share the same graphic object. For a control item[94], it allows to reuse its control after the item deletion.

Incremental printing

TRVPrint[479] and TRVReportHelper[503] components have new properties: MinPrintedItemNo[516] and MaxPrintedItemNo[515]. If these properties define a subrange of items, only this subrange (and only the pages containing it) is printed.

These properties allow implementing an incremental printing: when the next portion of the document is ready, it can be printed below the previously printed fragment. This feature can be used for printing accounting journals, logs, etc.

Editing

Triple click selects paragraph.

Horizontal coordinate is kept when moving the caret up or down with the keyboard.

When selecting from bottom to top, CurTextStyleNo[290] is defined as a style of the last selected text (to the right of the caret, not to the left as usual).

The methods ApplyStyleConversion[299] and ApplyParaStyleConversion[298] have the new parameter (Recursive) allowing to prevent processing of table cells.

Drawing and formatting

Improved table layout algorithm.

New option in RVVisibleSpecialCharacters[665]: *rvscTab*. RVParagraphMarks[662] global variable allows defining how paragraph breaks and line breaks are displayed in "show special characters" mode.

New property TRVStyle.DisabledFontColor[389] allows change text color in disabled controls.

Calls of TRVStyle.OnDrawTextBack[416] are optimized; now it is called only once for one text item on the line (previously, for justified paragraphs, it was called separately for each word)

TCustomRVPrint.GetPageNo[518] returns the page index for the given position in the document.

Saving and loading

Improved saving of footnotes[104] and endnotes[103] in plain text files.

Smarter replacing space characters to   when saving HTML.

New text properties for TRichView.DocParameters[140]:

- Author[277],
- Title[280],
- Comments[278]

Hypertext

rvoNoReadOnlyJumps option from EditorOptions[291] disallows automatic switching to a hypertext mode when TRichViewEdit becomes read-only[294].

1.11.6 New in version 12

Compatibility issues

Assignment to PageBreaksBeforeItems[149] requires reformatting even if the given item starts a paragraph (because it clears text flow around side-aligned items).

OutlineLevel[454] property allows saving headings in HTML and RTF, so be careful if you assumed that saved HTML and RTF files do not have headings.

Some existing code may assume that the text searching functions can select a substring only in a single text item[88]. While the new behavior (searching in multiple text items) requires specifying a new option in the parameter of SearchText methods explicitly, GetRVSearchOptions[616] and GetRVESearchOptions[616] functions always return this new option.

Left- and right-aligned objects

Two new values are available for TRVVAlign[648] type: *rvvaLeft* and *rvvaRight*. They can be used for all non-text items, except for tables[98], *breaks*[93], tabs[89], list markers[99], footnotes[104] and endnotes[103].

New methods are available for accessing vertical alignment:

- TRichView.GetItemVAlign[196],
- TRichView.SetItemVAlign[233],
- TRichViewEdit.GetCurrentItemVAlign[314],
- TRichViewEdit.SetItemVAlignEd[353],
- TRichViewEdit.SetCurrentItemVAlign[353].

Properties and methods for controlling text flow around side-aligned images:

- TRichView.ClearLeft[138],
- TRichView.ClearRight[138],
- TRichViewEdit.ClearTextFlow[305].

Showing invisible characters

New options for RVVisibleSpecialCharacters[665] allowing to display/hide lines for floating objects, paragraph directions, printing options and outline levels[454].

New colors can be defined in TRVStyle:

- FloatingLineColor[390],
- SpecialCharactersColor[402].

Headings

New property in paragraph style: OutlineLevel[454] defines headings.

Other

New option of text searching (*rvsroMultiItem* for TRichView.SearchText[222], *rvseoMultiItem* for TRichViewEdit.SearchText[336]) allows the searched string matching substrings of several text items.

New event in TRVStyle: OnAfterApplyStyle[410].

New event in TRichView: OnGetItemCursor[240]

New property in TRVLabelItemInfo: RemoveInternalLeading[581]

New methods in TRVOfficeConverter: IsValidImporter[502] allows checking a file before importing.

New values in TRichView.RVFOptions[152] and RVFWarnings[155] (*rvfoIgnoreUnknownCtrlProperties*, *rvfwUnknownCtrlProperties*)

1.11.7 New in version 11

The main feature of this update is compatibility with Delphi and C++Builder 2009.

Compatibility issues

New string types

New string types are declared in RVTypes.pas: TRVAnsiString[623], TRVUnicodeString[647], TRVRawByteString[638]. These string types are used instead of String and WideString in many methods, properties and events.

TRVAnsiString is used for parameters and variables containing ANSI strings.

TRVUnicodeString is used for parameters and variables containing Unicode strings.

TRVRawByteString is used for parameters and variables containing either ANSI or “raw Unicode” strings.

The standard String type is used for parameters and variables containing ANSI strings in older versions of Delphi/C++Builder, and Unicode strings for Delphi/C++Builder 2009.

Changed parameters of events

In the following events, type of some parameters was changed from String to TRVRawByteString[638]: OnDrawStyleText[415], OnReadHyperlink[248], OnItemAction[244], OnSaveItemToFile[261], OnRVDblClick[250], OnRVRightClick[255], OnItemTextEdit[360]. In existing projects, change types of parameters in these events manually.

In OnSaveComponentToFile[256] event, it is assumed that OutStr parameter is ANSI string (for Delphi/C++Builder versions prior to 2009). When saving to UTF-8 HTML files, it is converted to UTF-8 automatically. In previous versions of TRichView, programmers need to return UTF-8 string in this case.

Removing (renaming) "not recommended for Unicode applications" methods

In older versions of TRichView, there were a group of methods marked as "not recommended for Unicode applications". In TRichView version 11, these all these methods were renamed, R is added to the end of names

of these methods. For example, "GetItemText" was renamed to GetItemTextR. These methods have TRVRawByteString[638] parameters. These methods are not described in the help file any more (you will not see "not recommended for Unicode applications" remark in this help file any more). Instead of these methods, a new group of methods is introduced: methods working with String type (Unicode for Delphi/C++Builder 2009, ANSI for older versions).

For example, GetItemTextA always returns ANSI string, GetItemTextW always returns Unicode string, GetItemText returns ANSI/Unicode string depending on Delphi version. The methods without -A and -W postfixes can be used to create projects portable to all versions of Delphi.

The affected methods are:

- AddNLTag[167] (and its versions);
- GetWordAt[205];
- GetItemText[195];
- SetItemText[232];
- GetCurrentItemText[313];
- SetItemTextEd[352];
- SetCurrentItemText[347].

The same naming convention is applied to the methods that previously had two versions (ANSI and Unicode):

- CopyText[178];
- GetSelText[204];
- InsertStringTag[326];
- InsertText[327];
- PasteText[331];
- SearchText[222].

The following methods have only String (ANSI/Unicode string depending on Delphi version) version:

- AddFmt[163];
- AddItem[166];
- InsertItem[320];
- GetTextInfo[204];
- GetCurrentTextInfo[316].

The following method is removed (only -R version): AddTextBlockNL.

These changes break some existing low level code, especially the changes in GetItemText and SetItemText. Specifically, in the following examples GetItemText/SetItemText must be changed to GetItemTextR/ SetItemTextR:

- conversion to/from Unicode (ConvertToUnicode)[64];
- removing text formatting[673].

Delphi and C++Builder 2009

In the new versions of Delphi/C++Builder, String type is Unicode by default.

Many properties and parameters in TRichView become Unicode in Delphi/C++Builder 2009:

- *tag*[34] strings[34] (in tags-are-PChars mode);
- extra item properties[630];
- names of *checkpoints*[30];
- visible text in labels[101], numbered sequences[102], endnotes[103], footnotes[104];
- text in list markers[99] (not only bullets, but all numbering);
- live spelling[69] interface;
- hypertext targets (OnReadHyperlink[248], OnWriteHyperlink[248]);
- and others.

Default (initial) values of the following properties are changed in Delphi/C++Builder 2009:

- Unicode[446] property of text style (from False to True);
- TRichView.RTFReadProperties[151].UnicodeMode[274] (from *rvruNoUnicode* to *rvruOnlyUnicode*);
- TRichView.Options[146] (*rvoAutoCopyUnicodeText* is included, *rvoAutoCopyText* is excluded).

When saving text styles (in RVF files or Delphi forms) in older versions of Delphi/C++Builder, only non-default value (True) of Unicode[446] property of text style is saved. When saving text styles (in RVF files[59] or Delphi forms) in Delphi/C++Builder 2009, value of Unicode[446] property is always saved, default or not. The main consequence is the following: when loading forms/RVF files with styles saved by older versions of Delphi/C++Builder in Delphi/C++Builder 2009+, Unicode property of all text styles become True. For RVF files, all text in text items is converted to Unicode automatically.

Events parameters of TRVRawByteString type (OnDrawStyleText[415], OnReadHyperlink[248], OnItemAction[244], OnSaveItemToFile[261], OnRVDblClick[250], OnRVRightClick[255], OnItemTextEdit[360]) must be converted to String before using. For example, for OnRVDblClick[250] event:

```
// Example for Delphi 2009 or newer!!!
procedure TForm1.RichView1RVDblClick(Sender: TCustomRichView[128];
  ClickedWord: TRVRawByteString[638]; Style: Integer);
var s: String;
begin
  if (Style>=0) and Sender.Style[156].TextStyles[404][Style].Unicode[446] then
     s := RVU_RawUnicodeToWideString[620](ClickedWord) // converting from "raw Unicode" to U
  else
     s := RVU_RawUnicodeToWideString[620](  // converting from ANSI to UnicodeString
      RVU_AnsiToUnicode[620](Sender.RVData[151].GetStyleCodePage(Style), ClickedWord));
  Panel1.Caption := s;
end;
```

New methods, properties and events

New TCustomRichView.OnAddStyle[238] event.

New NoVScroll[511] property of TRichView.

New options for line spacing[452] ("at least" and "exactly" options).

New in saving and loading

RTF:

New option for saving formatted header and footer in RTF: include *rvrtfSaveHeaderFooter* in RTFOptions[150].

New properties for RTFReadProperties[151]: CharsetForUnicode[269] and UseCharsetForUnicode[275].

RVF[59]:

Applications compiled in Delphi/C++Builder 2009 save RVF version 1.3.1, applications compiled in the older versions of Delphi saves RVF version 1.3 (previous version of RVF). RVF version 1.3.1 stores many strings (checkpoints names, DocProperties, tags, and others) in UTF-8 encoding.

Other

Special support for non-breaking hyphens in Unicode text, better table layout algorithm, copying multicell selection as text (in addition to RVF), making multicell selection with keyboard.

1.11.8 New in version 10

Compatibility issues

The following changes may affect existing applications.

HTML

- HTML tag names, color constants, CSS names are saved in lower case;
- SaveHTMLEx[217] does not use the following attributes for cells and tables any more: bordercolor, bordercolorlight, bordercolordark, background, bgcolor. CSS style is used instead. These attributes are not

standard.
- OnSaveItemToFile[261] is called with Unicode parameter = True when saving UTF-8 HTML files ("raw Unicode" text is expected on output).

Tables

- Table layout algorithm: when there is not enough space for columns having widths specified in pixels, they are decreased proportionally (previously, only the rightmost columns were decreased)
- By default, table grid is not shown in read-only editors any more (it also affects TBDRichViewEdit when its dataset is not in the edit mode).
- In editor, Tab key in the last table cell adds a new row; set the global variable RichViewTableAutoAddRow[657] to *False* to disable.

Editing

- BeginUpdate[302] and EndUpdate[307] have new functionality.
- Simple version GetWordAt[205] always returns ANSI string.
- Paste[328] can paste graphic files in CF_HDROP format.

RTF

- bookmarks are read from RTF as checkpoints[30].
- Local links are converted automatically when reading and writing RTF.
- Sub/superscripts and VShift[443] are saved and loaded to RTF differently.

Other

- meaning of ItemNo parameter in OnControlAction[239] is changed.
- RVF insertion methods ignore background.

Main new features

- Animation of pictures[56];
- "Smart popups"[368];
- page breaks inside table cells.

New item types[29]

- labels[101];
- sequences[102];
- footnotes[104];
- endnotes[103];
- references to parent notes[108].

New in saving and loading

HTML:

New options in Options parameter of SaveHTML[216] and SaveHTMLEx[217]:

- rvsoXHTML[639] – if included, XHTML is saved instead of HTML;
- rvsoUTF8[639] – if included, HTML/XHTML is saved in UTF-8 encoding.

You can choose between using <p> and [652].

Font families are stored in CSS for the most frequently used fonts. For example, if font name is 'Arial', the string *'Arial', 'Helvetica', sans-serif* will be saved.

RTF:

You can specify class representing PNG images using the function RegisterPngGraphic. After that, TRichView will be able to write and read PNG images in RTF.

You can save page parameters using DocParameters[140] property.

Options[150] for saving smaller RTF files: *rvrtfSavePicturesBinary*, *rvrtfPNGInsteadOfBitmap*.

RTF bookmarks are read as *checkpoints*[30]. When exporting, links started from '#' are converted to links to checkpoints. When importing, links to checkpoints are converted to links started from '#'.

New properties for TCustomRichView.RTFReadProperties 151:

- StoreImagesFileNames 273 allows storing file names of external images;
- BasePathLinks 269 specifies whether to add path to file in hyperlinks;
- IgnoreBookmarks 271 allows ignoring bookmarks;
- IgnoreNotes 271 allows ignoring footnotes and endnotes;
- IgnoreSequences 272 allows ignoring numbered sequences;
- IgnoreTables 272 allows ignoring tables;
- ReadDocParameters 273 allows reading DocParameters;
- UseSingleCellPadding 275 sets mode for loading tables cell padding.

Text:

- You can allow saving explicit page breaks in text files/streams. Set RichViewSavePageBreaksInText 652 typed constant to *True*.

Other:

- LoadFromStream 208 loads document in RVF, RTF or text formats (autodetect).

New in DB controls

- TDBRichView 373 and TDBRichViewEdit 377 can be placed in TDBCtrlGrid.
- TDBRichView 373 and TDBRichViewEdit 377 can update standard DB actions.
- TDBRichView 373.OnLoadCustomFormat 376, TDBRichViewEdit 377.OnLoadCustomFormat 381 and OnSaveCustomFormat 382 events allow loading and saving data to fields in custom formats.
- Special support for Unicode memo fields.

New in drag&drop 55

Low level events:

- OnOleDragEnter 361;
- OnOleDragOver 362;
- OnOleDragLeave 362;
- OnOleDrop 363.

New in Unicode processing

- Support for the word joiner Unicode character (code $2060). This character is invisible and it disallows line breaking before and after it. A special support is required because almost no font has a glyph for this character (it works properly only if inserted in the middle of text item).
- WM_UNICHAR message is processed, if the current text is Unicode.
- TCustomRichView.SearchTextW 222 and TCustomRichViewEdit.SearchTextW 336 allows searching for WideString.

New in layout and formatting

- If you set RichViewShowGhostSpaces 656 typed constant to *True*, spaces hidden between lines will be shown if *rvoShowSpecialCharacters* is in TCustomRichView.Options 146.
- You can define which characters to show in ShowSpecialCharacters 146 mode. They are listed in RVVisibleSpecialCharacters 665 typed constant.
- New possible values for TCustomRichView.BackgroundStyle 136.
- New property TCustomRichView.MaxLength 145 – maximal count of characters per line.
- New property TCustomRichView.WordWrap 159 allows to disable word wrapping.
- If you set RichViewWrapAnywhere typed constant to *True*, text can be wrapped in any place (this constant was removed in version 13 682).

New text attributes

- New text Protection 439 options: *rvprStyleSplitProtect*, *rvprSticking2*, *rvprSticking3*.
- New property HoverEffects 437 allows hyperlinks underlining.
- New properties UnderlineType 442, UnderlineColor 442 and HoverUnderlineColor 437 allow to customize underlines.
- New property SubSuperScriptType 441 allows implementing subscripts and superscripts.

New in printing

- TRVReportHelper[503] and TRVPrint[479] do not require parent control any more. They can be placed on data module or created without owner.
- TRVPrintPreview[384] displays full paper size, including areas where printer cannot print (see WinAPI function GetDeviceCaps, PHYSICAL*** constants).
- TRVPrintPreview.PrintableAreaPen[523] property. If you set its Style<>*psClear*, RVPrintPreview shows borders of printable area (this rectangle was displayed as a full page in previous versions)
- TRVPrint.FixMarginsMode[484] property.

New in item properties

- *rvepVisible* item property[627] allows to hide controls[94];
- *rvepResizable* item property[627] works for pictures[90] too;
- *rvespAlt* item property[630] can be applied to *bullets*[96] and *hotspots*[97] as well as to pictures[90] and *hot-pictures*[90];
- new values for VAlign[648] property of items: *rvvaAbsTop*, *rvvaAbsBottom*, *rvvaAbsMiddle*.
- new styles[624] for *breaks*[93] (horizontal lines): dashed and dotted; all types of *breaks* are drawn without rounded corners.

New in *checkpoints*[30]

New set of editing methods working with checkpoint exactly at the position of caret:

- InsertCheckpoint[318];
- GetCheckpointAtCaret[307];
- RemoveCheckpointAtCaret[333].

New in tables[98]

- Table cells can be printed on several pages (previously, page breaks were possible only between rows).
- New option for table.Options[540]: *rvtoNoCellSelect*. If included, multicell selection by mouse is not allowed.
- New property table.VisibleBorders[544] allows to hide some sides of table border. Direct assignment to this property cannot be undone/redone, use table.SetTableVisibleBorders[563] for that.
- Table grid is not shown in read-only editors any more. More exactly, it is shown using global variable RichViewTableGridStyle2 (default value: *psClear*). The same pen style is used for drawing table grid in viewer.
- Copying multicell selection as a table fragment (in RVF[59] format); special pasting is not implemented yet, pasted as a new table.
- Adding new table row when user presses **Tab** in the last cell (set the global variable RichViewTableAutoAddRow[657] to *False* to disable this feature).
- New properties RowCount[543], ColCount[538].
- New properties CellHPadding[536], CellVPadding[537].
- New property Cell[567].Hint[571].
- New property Row[577].PageBreakBefore[578].
- New methods SetCellBackgroundImageFileName[559] and SetCellHint[561].
- Improved table layout algorithm.
- better SplitSelectedCellsVertically[563] and SplitSelectedCellsHorizontally[563] implementations.

New in RVF[59]

If image was not returned in OnRVFPictureNeeded[252] event, this event is called the second time with the value of *rvespImageFileName*[630] property in the Name parameter (if value of this property was stored for this item). (update: in TRichView 14, OnRVFPictureNeeded is reworked and it is not called twice any more)

Custom Caret

You can draw your own caret instead of the system one. It is activated CustomCaretInterval[290], measured in

OnMeasureCustomCaret 360 event, drawn in OnDrawCustomCaret 358 event.

Other

- The following Windows messages are processed by TCustomRichView: EM_GETSEL, EM_SETSEL, EM_GETTEXTRANGE. If you do not need these messages, you can define RVDONOTUSELINEARPOSITIONS in RV_Defs.inc. It slightly reduces exe-file size.
- Pasting graphic files in CF_HDROP format, see PasteGraphicFile 329.
- New characters are included in the default value of TCustomRichView.Delimiters 140.
- meaning of TCustomRichViewEdit's BeginUpdate 302 and EndUpdate 307 was changed.
- New option for TCustomRichView.SearchText 222: *rvsroFromStart*.
- Changes in TCustomRichView.GetWordAt 205.
- Ability to delete unused styles in multiple documents, see TRVDeleteUnusedStylesData 612.
- Much faster reformatting, especially if *rvoFastFormatting* is included in TCustomRichView.Options 146 (default). Much faster loading RTF files. Much faster copying to the Clipboard.
- Function RVIsCustomURL 662.
- Ctrl+[←] and Ctrl+[↓] scroll the document down/up instead of moving caret.
- (in Unicode mode in WinNT/2000/XP/Vista) special support for typing in Kazakh, Tatar, Mongolian, Azerbaijani (Cyrillic & Latin), Uzbek, Kyrgyz, Serbian.
- New "hidden" parameters in TCustomRichView.OnControlAction 239.

1.11.9 New in version 1.9

Compatibility issues

- TRVStyle.CurrentItemColor 388 is used for drawing frame around the current item instead of TRVStyle.HoverColor 393.
- ApplyTextStyle 300, ApplyParaStyle 297, ApplyStyleConversion 299, ApplyParaStyleConversion 298 are applied to cells of all selected tables.
- OnSaveImage2 260 event: ImageSaveNo is declared as a var-parameter (it does not make sense otherwise)
- RTF export: only Standard 431 styles can be exported as style sheet.
- Default value of RVStyle.SpacesInTab 402 is changed from 8 to 0.
- Default value of TRichViewEdit.EditorOptions 291 is changed from [rvoClearTagOnStyleApp] to [rvoWantTabs, rvoCtrlJumps].
- New characters are added to default value of Delimiters 140 property.

New: tab stops

- Tabs 455 – properties of paragraph style defining tab stops.
- TRVStyle.DefTabWidth 388 defines default distance between tab stops.
- TRVStyle.SpacesInTab 402: if equals to 0, methods for adding/inserting multiline text, RTF, and Tab key inserts tabulator 89 (item of special type). If this property has positive value, all these methods work like before (inserting several spaces in place of tabs).
- TRichView.AddTab 170, TRichViewEdit.InsertTab 326 adds/inserts tabulator 89

New: Live spelling check

See overview topic 69

New in TRichView

- VAlign 159 property, ClientToDocument 176 method;

- *rvoShowSpecialCharacters* in Options[146];
- BeginOleDrag[174] – allows drag&drop of inserted controls;
- Reformat[214] method – formatting method that keeps selection;
- AddTextNLA[171] method;
- LoadTextFromStream, LoadTextFromStreamW[213];
- Events: OnItemHint[246], OnProgress[248].

New item properties

- additional string properties[630]: hint, alt text, image file name; a set of methods supporting them: TRichView's GetItemExtraStrProperty[193] and SetItemExtraStrProperty[231], TRichViewEdit's GetCurrentItemExtraStrProperty[313], SetCurrentItemExtraStrProperty[346] and SetItemExtraStrPropertyEd[351] (this set of methods is similar to methods working with additional integer properties[627]);
- *rveoNoHTMLImagesSize* additional integer property[627]

Printing

Added: *rvpaoKeepLinesTogether* and *rvpaoKeepWithNext* Options[453] for paragraph style.

Tables

- *rvtoIgnoreContentWidth* and *rvtoIgnoreContentHeight* options for TRVTableItemInfo[528].Options[540];
- AssignProperties[547] method;
- better table layout algorithm.

RTF

- TRichView.RTFReadProperties[268].ExtractMetafileBitmaps[271] – allows extracting bitmaps wrapped in metafiles.
- output RTF file size is greatly reduced;
- a number of small improvements in RTF import and export.

HTML export

- SaveHTMLEx[217] is modified to conform "HTML 4.01 Transitional" standard, see http://validator.w3.org/;
- improvements in CSS export;
- SaveHTMLEx[217] saves characters of "Symbol" font as HTML entities (like α);
- better saving of RTF-like tables (with CellHSpacing[536]=CellVSpacing[537]=-1 to simulate 1-pixel width border) with SaveHTMLEx[217], or SaveHTML[216] with *rvsoForceNonTextCSS* in the Options parameter;
- ImagesPrefix parameter of SaveHTML[216]/SaveHTMLEx[217] can contain a full path (determined by the presence of ':').

TRVOfficeConverter[496]

- added: PreviewMode[499] property.

TRVReportHelper[503]

- added: OnDrawCheckpoint[507] event

Database controls

- OnLoadDocument 382 event for TDBRichViewEdit.
- OnNewDocument 377 and OnLoadDocument 376 for TDBRichView.

Other improvements

- TRVStyle.CurrentItemColor 388.
- *rvoNoCaretHighlightJumps* in EditorOptions 291;
- faster redrawing on typing;
- better processing of bidirected text 69;
- ApplyTextStyle 300, ApplyParaStyle 297, ApplyStyleConversion 299, ApplyParaStyleConversion 298 are applied to cells of all selected tables;
- all TRichViewEdit methods for inserting one item 52 return boolean value: *True* if the item was successfully inserted (insertion can fail because of protection). You can use SetCurrent*** methods to set additional properties of this item after the insertion.
- support for soft (optional) hyphens (WideChar($AD)) and zero-width spaces (WideChar($200B)) in Unicode text. Normally soft hyphen is invisible. It is displayed as a hyphen only if line breaks is after it.If *rvoShowSpecialCharacters* is in Options, 146 it is shown as "not" sign (WideChar($AC)), like in MS Word. Unlike spaces and paragraph marks, soft hyphens change their widths when switching normal/show-special-chars modes. So call Reformat 214 when switching these modes. Processing of these characters may slow down formatting of Unicode text. If you do not need them, activate RVDONOTUSESOFTHYPHENS in RV_Defs.inc Zero-width spaces are processed correctly even if this character is not present in the font.
- RVF (RichView Format) forward compatibility improved.

1.11.10 New in version 1.8

Compatibility issues

Changes that can affect existing code:

- OnHTMLSaveImage 241 is called for tables to save background image;
- OnHTMLSaveImage 241 is called to save cell background image; in this case, parameter ItemNo = -1; you can distinguish saving of document background and cell background by RVData parameter.
- MaxTextWidth 146 property does not include left and right margins any more, so now it works according to the help file 46;
- parameters are changed in OnChanging 357 event (should not be a problem because this event is rarely used);
- table.OnDrawBorder 565 event has two new parameters: Row and Col;
- changes in *rvteoHTMLCode* and *rvteoRTFCode* (Options 438 of text style);
- TRichView does not use font external leading any more (it may add extra spacing between lines), so its line spacing just like in TRichViewEdit now.
- default selection mode: select by words 399;
- RVF version number is changed from 1.2 to 1.3 (see RVF specification 77); new version of RVF is not completely compatible with the old version (new version of TRichView can read old RVF; old versions of TRichView may not read new RVF);
- *rvprSticking* 439 protection option meaning was changed (split into 3 rvprStick*** options).

Drag&Drop

See overview 55.

Background images for tables and cells

- table.BackgroundImage 530, BackgroundStyle 531;
- cell.BackgroundImage 567, BackgroundStyle 568.

Resizing pictures and controls with mouse

By default, all pictures 90 and hot-pictures 93 can be resized with the mouse (except for TIcon-s).

By default, controls 94 cannot be resized. But you can explicitly allow resizing for the given control, if you set its *rvepResizable* 627 property to non-zero value. See also SelectControl 223 method.

You can disallow resizing in the given RichViewEdit by including *rvoNoImageResize* in its EditorOptions 291 property.

Extra items' integer properties 627

- *rvepSpacing* – spacing around the item (image or control);
- *rvepResizable* allows resizing the given control with the mouse;
- *rvepDeleteProtect* protects the non-text item from deletion in editing operations.

Printing and preview

- ColorMode 514 property for TRVPrint 479 and TRVReportHelper 503: allows black and white or grayscale printing/previewing for color documents;
- much more accurate print preview 384;
- new properties of TRVPrintPreview: ClickMode 521, Color 521, PageBorderColor 522, PageBorderWidth 522, ShadowColor 523, ShadowWidth 523.

Selecting:

- new line selection mode, see TRVStyle.LineSelectCursor 394
- defining visual appearance of selection, see TRVStyle.SelectionStyle 399
- defining selection mode, see TRVStyle.SelectionMode 399

RVF 59

- tags can contain space characters;
- in text-mode RVF, TRichView saves text RVF files even if the document contains Unicode text (text files can be edited in text editors)
- ability to save arbitrary additional text strings (DocProperties 141).

HTML export

- by default, all images are saved in HTMLs as Jpegs. This is obviously undesirable if images already have formats recognizable by web browsers (such as Gif or Png). Now you can specify graphic formats that must not be converted to Jpeg on HTML export, using RV_RegisterHTMLGraphicFormat procedure.
- OnSaveHTMLExtra 259 event now allows to insert HTML code before the </body> tag;
- new OnSaveParaToHTML 263, OnSaveImage2 260, OnSaveItemToFile 261 events;
- new options 639 for HTML export;

- OnWriteHyperlink 267 event supersedes OnURLNeeded.

RTF export

- OnWriteHyperlink 267 event supersedes OnURLNeeded;
- new OnSaveItemToFile 261 event.

RTF import

- import of DDB (device dependent bitmaps);
- using some advanced RTF features when importing images from RTF saved by MS Word 2000+.
- more special RTF characters are recognized
- new event: OnImportPicture 242

DB controls

- new event: TDBRichViewEdit.OnNewDocument 382;
- new property: TDBRichViewEdit.IgnoreEscape 380.

Other improvements

- TRVReportHelper.OnDrawHyperlink 508 event;
- default text styles for paragraph styles (TParaInfo.DefStyleNo 457);
- improvements in processing of bidirected text 69;
- new action in OnItemAction 244 event: *rviaInserted*;
- faster redrawing on editing;
- previously reserved break styles 624 are implemented;
- RefreshListMarkers 214 method.

1.11.11 New in version 1.7

Compatibility issues

- TCustomRichView.DeleteLines is renamed to DeleteItems 179.
- If you assign non-default values to extra item properties 627, saved RVF files will not be opened with applications that use older version of RichView.
- OnDrawPageBreak 413 event has additional parameter. If you already use this event, you must change declaration of event handler manually.
- All references to TImageList were replaced with TCustomImageList. Especially important: parameter of OnRVFImageListNeeded 252 was changed; if you already use this event, you must change declaration of event handler manually;
- FirstIndent 451 is not applied to line starts inside paragraphs (Shift+Enter).
- DeleteUnusedStyles 181 has a new parameter (deleting list styles).
- GetWordAt 205 parameters are changed.
- Accessing invalid item in RVStyle.TextStyles and ParaStyles does not cause "List index is out of bounds" error any more.
- OnHTMLSaveImage 241 is called for background bitmap 136; this may affect old code that assumes ItemNo is

always an index of item.

- Declarations of some types are moved in other files.
- Format of CSS was slightly changed to support list styles.

The main new feature: paragraph lists (bullets and numbering)

- TRVStyle.ListStyles 396

Methods of TCustomRichView:

- SetListMarkerInfo 233,
- GetListMarkerInfo 199.

Methods of TCustomRichViewEdit:

- ApplyListStyle 297,
- RemoveLists 334,
- ChangeListLevels 304.

New options 639 for HTML export – *rvsoMarkersAsText*.

New parameter in DeleteUnusedStyles 181 (deleting list styles)

Extra integer properties for items (stretching, transparent color for bitmaps, etc)

Methods of TCustomRichView:

- SetItemExtraIntProperty 230 – changes value of property (without document reformatting);
- GetItemExtraIntProperty 193 – returns value of property.

Methods of TCustomRichViewEdit:

- SetItemExtraIntPropertyEd 350 – changes value of property, updates the document;
- SetCurrentItemExtraIntProperty 345 – the same for item at the position of caret;
- GetCurrentItemExtraIntProperty 312 – returns value of property of item at the position of caret.

Soft page breaks (preliminary implementation)

Methods of TCustomRichView:

- AssignSoftPageBreaks 174, ClearSoftPageBreaks 176 - displays/hides soft page breaks

Property of TRVStyle

- SoftPageBreakColor 401

Other improvements in TCustomRichView

New methods:

- DeleteParas 180 – deletes the specified paragraphs, updates the document;
- GetLineNo 198 – returns line for the specified position;
- GetWordAt 205 (parameters are changed);
- GetItemAt 190 – returns item in the given coordinates.

New events:

- OnItemAction 244 – occurs on operations with item;
- OnPaint 247.

New properties:

- RTFReadProperties.ConvertHighlight 270, LineBreaksAsParagraphs 272.

New Options 639 for HTML export: *rvsoUseCheckpointsNames.*

New Options 152 for RVF: *rvfoSaveLayout*, *rvfoLoadLayout*.

Events:

- OnSaveHTMLExtra 259, OnSaveRTFExtra 263 – allow saving additional information in HTML and RTF.

Other improvements in TCustomRichViewEdit

Methods:

- ApplyParaStyleConversion 298 (and OnParaStyleConversion 364 event) – implements commands like "change paragraph alignment", "increase indent", etc.
- InsertText, InsertTextW 327 have additional optional parameter;
- GetCurrentLineCol 314 – returns line and column at the caret position;
- ConvertToPicture 305, ConvertToHotPicture 305 – convert picture to *hot-picture* and vice versa

Properties:

- Modified 293: Boolean

New EditorOption 291 – *rvoDoNotWantShiftReturns*.

Other improvements in TRVStyle 385

- InvalidPicture 393 – picture that used instead of invalid or missed pictures;
- InvalidItem 418 for TextStyles, InvalidItem 423 for ParaStyles – are returned when accessing item which is out of index range.

New properties of text style (TFontInfo 444):

- Options 438 – affects HTML and RTF export

New properties of paragraph style (TParaInfo 448):

- new protection Option 453 – *rvpaoDoNotWantReturns*.

Improvements in TDBRichViewEdit 377

- AutoDeleteUnusedStyles 378

Improvements in TRVOfficeConverter 496

- ExcludeHTMLImportConverter, ExcludeHTMLExportConverter 498;
- ExtensionsInFilter 499;
- ErrorCode 497 – returns error code for the last import/export operation.

Improvements in TRVPrint 479

- OnPrintComponent 519 is optional since this version;
- ClipMargins 484 property – disallows printing on margins;
- headers and footers: TRVPrint.SetHeader 493, SetFooter 493, HeaderYMM, FooterYMM.
- reading headers and footers from RTF: TRichView.RTFReadProperties.SetHeader 276, SetFooter 276, HeaderYMM, FooterYMM;
- OnPagePostpaint 495;
- ColorMode 514 – allows black&white and grayscale printing and preview;

Improvements in all visual controls of the package

- UseXPThemes 511 - allows using WindowsXP themes (visual styles)

Improvements in tables 111

Global variable RichViewTableGridStyle: TPenStyle – drawing invisible table borders.

Methods:

- SetCellVisibleBorders 562 – sets VisibleBorders property for the cell.

Properties:

- HeadingRowCount 539 – defines table header;
- new Option 540 – *rvtoRTFAllowAutofit*.

Events:

- OnCellEditing 564 – allows to prevent cell editing;
- OnDrawBorder 565 – custom drawing for cell borders.

New help topics

- Valid Documents 74
- Custom Drawing 76
- Examples for DBRichViewEdit 377
- New overview: Item Types 86
- More examples and cross references.

Miscellaneous

Better RTF import (including import of inserted symbols, Unicode paired quotes and dashes), better IME support, better Unicode text processing, faster formatting, and so on.

Packages for D7 and CB6.

1.11.12 New in version 1.6

Compatibility issues

- TRVPrint.OnPrintComponent 519 event has changed type of Sender parameter (from TRVPrint to TCustomRVPrint). If you use this event in your existing projects, change type of this parameter in your handlers for this event.
- Parameters added in TRichView.DeleteUnusedStyles 181.

Changes in TRVPrint

- new: TRVPrint.TransparentBackground 517
- new: TRVPrint.GetFirstItemOnPage 517
- new: TRVPrint.MirrorMargins 487

Changes in tables

- Vertical alignment of cells (cells' VAlign 575 and rows' VAlign 578, SetCellVAlign 562 and SetRowVAlign 562 of table).
- BorderColor 569 and BorderLightColor 570 of cells allow to override default values (see also SetCellBorderColor 560 and SetCellBorderLightColor 560).
- Ability to hide table grid in editor (new option for table Options 540 – *rvtoHideGridLines*).
- Ability to print table background in white (new option for table PrintOptions 542 – *rvtoWhiteBackground*).
- New table Options 540 – *rvtoCellBelowBorders* and *rvtoOverlappingCorners*.
- Tables with *rvtoEditing* excluded from Options now ignore commands for cells clearing and style applying.
- Hidden sides of borders can be shown as dotted gray line (if *rvtoHideGridLines* is not in table Options 540). Previous versions show grid only for zero-width borders.

New components

 TRVOfficeConverter[496] – allows using Microsoft Office converters

 TRVReportHelper[503] – reserved for future use.

HTML export

New options[639] – *rvsoImageSizes*, *rvsoForceNonTextCSS*.

RTF export

You can save hyperlinks in RTF, using OnURLNeeded[266] event.

New option[150] for saving RTF files – *rvrtfSaveEMFDefault*.

RTF import

- new: reading hyperlinks: OnReadHyperlink[248] event;
- reading tables (see also: RTFReadProperties.AutoHideTableGridLines[269]);
- reading external images (see important note for using third-party graphics formats[672]);
- new: ignoring hidden text (see RTFReadProperties.SkipHiddenText[273]).

Horizontal scrolling

new: HScrollPos[510], HScrollMax[509], OnHScrolled[513].

Visual effects and formatting

- new: TRVStyle.InactiveSelColor and InactiveSelTextColor[393];
- new: CharSpacing[435] property of text style.

DB Components

DB components can display RTF and plain text fields.

new: TDBRichViewEdit.FieldFormat[380] allows to edit RTF and text fields

Miscellaneous

- New component-editor (click RichView and choose "Settings..." in popup menu) allows to set properties related to updating styles.
- New methods of TRichViewEdit: InsertStringTag and InsertStringWTag[326].
- Parameters are added in TRichView.DeleteUnusedStyles[181].
- Essential styles can be marked with Standard[431] property.
- TRichView.SearchText[222] and TRichViewEdit.SearchText[336] now searches in table cells.
- Hypertext works automatically in read-only[294] editors (user does not need to press and hold Ctrl).
- Editor can hide caret in read-only mode (new option in EditorOptions[291] - *rvoHideReadOnlyCaret*).
- Methods for loading text files understand page-break characters (characters with code $0C).
- Methods for loading text files can be used for Unix text files (with #$0A characters separating lines).
- TRichViewEdit.SelectCurrentLine[337].

1.11.13 New in version 1.5

Compatibility issues

1. Changed types in events. Important (please read before installing the update!):

Types of parameters were changed in some events: "TRichView" was changed to "TCustomRichView", "TRichViewEdit" to "TCustomRichViewEdit".

After installing, open all forms containing RichView or RichViewEdit, and modify declarations of event handlers: change "TRichView" to "TCustomRichView", "TRichViewEdit" to "TCustomRichViewEdit". You must do it manually!

(This is one-time change that is required only for projects created with older version of the components)

For example, for TRichView.OnRVMouseDown

```
procedure TForm1.RichView1RVMouseDown(
                Sender: TRichView;
                Button: TMouseButton;
                Shift: TShiftState;
                ItemNo, X, Y: Integer);
begin
...
end;
```

must be changed to

```
procedure TForm1.RichView1RVMouseDown(
                Sender: TCustomRichView;
                Button: TMouseButton;
                Shift: TShiftState;
                ItemNo, X, Y: Integer);
begin
...
end;
```

Sorry for inconvenience.

2. Changed component hierarchy

TRichViewEdit, TDBRichView, TDBRichViewEdit are not descendants of TRichView any more. All of them (and TRichView itself) are descendants of TCustomRichView.

TDBRichViewEdit is not a descendant of TRichViewEdit any more. Both them are descendants of TCustomRichViewEdit.

See new scheme of hierarchy 26.

3. New parameter

New parameter of SaveCSS 408/SaveCSSToStream 409. Please use empty set ([]) for it.

New feature: RTF Import

- TRichView.LoadRTF 209;
- TRichView.LoadRTFFromStream 210;
- TRichViewEdit.InsertRTFFromFileEd 323;
- TRichViewEdit.InsertRTFFromStreamEd 323;
- TRichViewEdit.PasteRTF 330;
- TRichViewEdit.CanPasteRTF 303;

- and property TRichView.RTFReadProperties 151.

New feature: saving text and paragraph styles in RVF

- New *rvfoSaveTextStyles* and *rvfoSaveParaStyles* RVFOptions 152
- TRichView.RVFTextStylesReadMode 154;
- TRichView.RVFParaStylesReadMode 154;
- Related: TRichView.DeleteUnusedStyles 181.

New feature: *hot pictures*

Hot pictures are the same as normal pictures, but they are also hypertext jumps.

New methods: AddHotPicture 163, AddHotPictureTag 163, InsertHotPicture 319.

Old methods (GetPictureInfo 201, SetPictureInfo 234, GetCurrentPictureInfo 315, SetCurrentPictureInfo 348, SetPictureInfoEd 354) now work with both pictures and *hot pictures*.

New properties for text and paragraph formatting:

- TFontInfo.StyleEx 441 (new option – *rvfsAllCaps*);
- TFontInfo.CharScale 434;
- TParaInfo.LineSpacing 452;
- TParaInfo.LineSpacingType 452;
- TParaInfo.Options 453.

New feature: Tab navigation through links and inserted controls

TRichView.TabNavigation 157 property;

TRichView.GetFocusedItem 188 method.

New feature: Proportional resizing of controls in documents

RichView now can resize controls depending on document width.

Let ItemNo - index of control item (rvsComponent).

```
var Item: TRVControlItemInfo;
  ...
Item := RichView.GetItem(ItemNo) as TRVControlItemInfo;
Item.PercentWidth := 50; // 50% of document width (minus paragraph indents)
```

Effect will be after reformatting 182.

Known problem: such controls may not work properly in paragraphs with word-wrapping turned off.

Miscellaneous

- OnCaretMove 356
- GetLastCheckpoint 198, GetPrevCheckpoint 202
- two new possible values of Direction parameter of OnCaretGetOut 356
- TRVStyle.ParaStyles.FindStyleWithAlignment 424 and FindSuchStyle 425
- "whole word" option for TRichView.SearchText 222 and TRichViewEdit.SearchText 336 (thanks to Louis Kessler)

Improved:

- support of third party graphic classes 671
- RTF export
- HTML export: SaveHTML 216 now retains right and justify alignments.

1.11.14 New in version 1.4

Compatibility issues

- TRichView.OnControlAction 239 declaration was changed. If you use this event, please add "var" to ctrl parameter of your event handler after rebuilding the package. Failing to do it causes memory access errors when running your applications.
- GetCurrent***Info/SetCurrent***Info methods of TRichViewEdit now differ from Get***Info(ItemNo,...)/ Set***InfoEd(ItemNo,...). "Current" methods can be used to work with items in cell-inplace editor in the same way as with items in main editor (programmer does not need to know if the item is in main editor or in sub-editor, he/she can always work with item at the position of caret). "Not-current" methods cannot be used to work with items in inplace editor, because such items cannot be addressed using only their indices.

The main new feature: tables

There is a new set of topics about them 111.

New methods

TRichView:
- AddNLATag 167, AddItem 166, GetItem 190, GetItemNo 194,
- GetJumpPointLocation 197
- GetItemCoords 191, GetItemClientCoords 191

TRichViewEdit:
- InsertItem 320,
- GetCurrentItem 311, GetCurrentItemEx 311
- SetItemTextEd 352,SetCurrentItemText 347, GetCurrentItemText 313,
- CanChange 302
- BeginItemModify 301, EndItemModify 307
- ResizeCurrentControl 335, AdjustControlPlacement2 296

TFontInfos:
- FindStyleWithCharset 420

New properties

- TParaInfo.NextParaNo 457
- TRVStyle.DefCodePage 388
- TRichView.HScrollVisible 510
- TRichView.RVData 151
- TRichView.InplaceEditor 510
- New option (rvoAutoSwitchLang) in TRichViewEdit.EditorOptions 291 (thanks to Pavel Zhelty)
- New states in TRVTextDrawStates 645

New events:

- TRichView.OnCopy 240
- TRichView.OnHTMLSaveImage 241

- TRichViewEdit.OnCaretGetOut 356
- New "hidden" parameters in TRVStyle.OnDrawStyleText 415 and OnDrawTextBack 416

Cursors

Two new cursors. Ability to use system default hypertext cursor 77

Miscellaneous

Special support for Unicode Hindi keyboard input (note: Hindi uses complex glyphs composed from several Unicode characters; such glyphs are not fully supported yet)

Known problems

- TRVStyle does not work properly in frame (Delphi 5+ and C++Builder 5+)
- wrong displaying of underline for Unicode text in Windows95 (WinNT/2000 is OK)

1.11.15 New in version 1.3

The main additions in version 1.3

- Undo/Redo 54 (many topics, this is the link to the overview);
- Unicode support 64 (many topics, this is the link to the overview);
- Events for custom drawing (several topics; this is the link to the list)
- TRichView.RTFOptions 150, CopyRTF 178, SaveRTF 219, SaveRTFToStream 220
- TRVPrint.OnPagePrepaint event
- TFontInfo.Protection 439 property
- TRVStyle.UseSound 404 property
- TRichView.AddTextBlockNL, SetItemText 232, GetItemText 195, SaveTextToStream, SaveTextToStreamW 221 methods;
- TRichView.OnControlAction 239 event;
- new options in TRichView.Options 146, TRichViewEdit.EditorOptions 291;
- new parameter in TRichView.LoadText 212;
- TFontInfo.IsEqual 444, TFontInfos.FindStyleWithFontStyle 421, FindStyleWithFontSize 421, FindStyleWithColor 420, FindStyleWithFontName 421, FindSuchStyle 422, FindStyleWithFont 420
- new global variables 652
- HTML export improved.

Compatibility issues

- declaration of some types were moved (usually from RichView.pas to either RVScroll.pas or RVStyle.pas; in some cases you will need to correct uses clause of units in older projects, if Delphi will not find some types).
- TRichView.LoadText has a new parameter.
- TFontInfo.SingleSymbols property removed; see TFontInfo.Protection 439.
- Older version of RichView will not be able to read RVF 59 with Unicode.

1.11.16 New in version 1.2

The main additions in version 1.2

- Borders and Background of Paragraphs 39;
- spacing between paragraphs (TParaInfo.SpaceBefore 455,TParaInfo.SpaceAfter 454) and ability to move items to the new line inside paragraphs (Shift+Enter in editor, also see TRichView.IsFromNewLine 207, TRichView.IsParaStart 208, TRichView.SetAddParagraphMode 225);
- ability to specify explicit page breaks (TRVStyle.PageBreakColor 398, TRichView.PageBreaksBeforeItems 149, TRichView.Options 146, TRichViewEdit.InsertPageBreak 321, TRichViewEdit.RemoveCurrentPageBreak 334);
- ability to change background color of text under mouse (TFontInfo.HoverBackColor 436);
- ability to save and load styles to registry (TRVStyle.LoadReg 408, TRVStyle.SaveReg 409, Delphi 4+ is required);
- ability to search upwards in RichView (TRichView.SearchText 222);
- ability to save background in RVF file or stream (optionally) (TRichView.RVFOptions 152, RVF Overview 59).

Compatibility issues

- default search direction in TRichView.SearchText 222 was changed;
- some keys for saving styles in ini-files were changed (SingleSymbolsN -> FontSingleSymbolsN, VShiftN -> FontVShiftN, NextStyleNoN -> FontNextStyleNoN, AlignmentN -> ParaAlignmentN);
- components do not use text HoverColor for drawing selected hypertext links any more; you can return old behavior turning on RVUSETEXTHOVERCOLORWITHSELECTED define;
- removed method TRVStyle.SaveCSSFile (replaced with TRVStyle.SaveCSSToStream 409);
- RVF format was changed. RichView v1.2 can read RVF files saved by old versions of RichView, but older versions can not read new RVF files.

New and updated topics (from version 1.1.3 to version 1.2)

New:

- TFontInfo.HoverBackColor 436
- TRVStyle.LoadReg 408
- TRVStyle.SaveReg 409

- TRVRect 615
- TRVBooleanRect 611
- TRVBorderStyle 623

- Borders and Background of Paragraphs 39

- TParaInfo.SpaceBefore 455
- TParaInfo.SpaceAfter 454
- TParaInfo.Border 451
- TParaInfo.Background 450

- TRVBorder 467
- TRVBorder Properties
- TRVBorder.BorderOffsets 467
- TRVBorder.Color 468
- TRVBorder.InternalWidth 468
- TRVBorder.Style 468
- TRVBorder.VisibleBorders 468

- TRVBorder.Width 468

- TRVBackgroundRect 466
- TRVBackgroundRect Properties
- TRVBackgroundRect.BorderOffsets 466
- TRVBackgroundRect.Color 467

- TRichView.IsFromNewLine 207
- TRichView.SetAddParagraphMode 225

- TRVStyle.PageBreakColor 398
- TRichView.PageBreaksBeforeItems 149
- TRichViewEdit.InsertPageBreak 321
- TRichViewEdit.RemoveCurrentPageBreak 334

- TRVStyle.SaveCSSToStream 409
- TRichView.SaveHTMLToStream 218
- TRichView.SaveHTMLToStreamEx 218

Updated (not including topics where only references were corrected):

- RichView Paragraphs 37, Building RichView Document 43
- Export of RichView contents to HTML 62, RichView Methods for Saving and Loading 58, Searching and Replacing 49,
- RVF Overview 59, RVF Specification 77
- TRVSearchOption 642
- TRichView.SearchText 222
- TRichViewEdit.SearchText 336
- TRichView.IsParaStart 208
- TRichView.Options 146
- TRichView.RVFOptions 152
- TRichView.SaveHTML 216
- TRichView.SaveHTMLEx 217
- TRVSaveOption 639
- TRichView.SavePicture 219

Removed:

- TRVStyle.SaveCSSFile

1.11.17 Compatibility with older versions

See compatibility issues for versions: 10 687, 1.9 691 1.8 693, 1.7 695 1.6 698 1.5 700, 1.4 702, 1.3 703, 1.2 704

For former users of freeware version

You can use this version of TRichView, TRVStyle, TRVPrint components in your projects started with older version.

But

- You lose all hypertext in text styles then you open old project with new TRVStyle; this is because there were only two hypertext styles in older versions (had indices *rvsJump1*, *rvsJump2*); now any text style can be hypertext, regardless of its index. In order to fix, set Jump 445=True, HoverColor 436 to TRVStyle.HoverColor 393 for hypertext items.
- There are some obsolete methods, but they still work. For example, all methods which in older versions added left-aligned items (AddFromNewLine), now add items with the 0th paragraph style; all methods which in older versions added center-aligned items (AddCenterLine) now add items with the 1st paragraph style. Initial values of paragraph styles 398: 0th paragraph is left aligned, 1th is centered by default.

- *Checkpoints* are not items any more, but some additional information associated with items. Add***Checkpoint methods still work: such method appends *checkpoint* that will be associated with the next appended item. Therefore, you now cannot add two or more *checkpoints* one by one without item between them. You can choose one of two options when compiling RichView source:
  - RichView raises exception when you try to add *checkpoint* after *checkpoint* (default mode);
  - RichView ignores the second *checkpoint*; Add***Checkpoint*** methods will return the index of previous added *checkpoint* in this case; this mode is turned on by defining RVALLOWCPBYCP conditional define.
- New version has no special design-time property editors for RVStyle.TextStyles
- For C++Builder users: some names of methods and properties are now written in another case. "CheckPoint" is "Checkpoint" now, "HotSpot" is "Hotspot" now.

Index

- A -

AcceptDragDropFormats 287
TRichViewEdit 287

AcceptPasteFormats 287
TRichViewEdit 287

ActualCurTextStyleNo 288
TCustomRichViewEdit 288

Add 167, 419, 424, 426, 472
TFontInfos 419
TParaInfos 424
TRichView 167
TRVListInfos 426
TRVListLevelCollection 472

AddBreak 160
TRichView 160

AddBreakEx 160
TRichView 160

AddBreakExTag 160
TRichView 160

AddBreakTag 160
TRichView 160

AddBullet 160
TRichView 160

AddBulletEx 160
TRichView 160

AddBulletExTag 160
TRichView 160

AddCenterLine 161
TRichView 161

AddCheckpoint 166
TRichView 166

AddCheckpointTag 166
TRichView 166

AddControl 162
TRichView 162

AddControlEx 162
TRichView 162

AddControlExTag 162
TRichView 162

AddFmt 163
TRichView 163

AddFont 419
TFontInfos 419

AddFontEx 419
TFontInfos 419

AddFrom 469
TRVTabInfos 469

AddFromNewLine 161
TRichView 161

AddHotPicture 163
TRichView 163

AddHotPictureTag 163
TRichView 163

AddHotspot 164
TRichView 164

AddHotspotEx 164
TRichView 164

AddHotspotExTag 164
TRichView 164

AddItem 166
TRichView 166

AddNamedCheckpoint 166
TRichView 166

AddNamedCheckpointEx 166
TRichView 166

AddNamedCheckpointExTag 166
TRichView 166

AddNL 167
TRichView 167

AddNLATag 167
TRichView 167

AddNLTag 167
TRichView 167

AddNLWTag 167
TRichView 167

AddPicture 169
TRichView 169

AddPictureEx 169
TRichView 169

AddPictureExTag 169
TRichView 169

AddTab 170
TRichView 170

AddTag 167
TRichView 167

AddText 171
TRichView 171

AddTextFromNewLine 171
TRichView 171

AddTextNL 171
TRichView 171

AddTextNLA 171
TRichView 171

AddTextNLW 171

AddTextNLW 171
TRichView 171
AddTextStyle 405
TRVStyle 405
AddXXX methods of TRichView 43
AdjustControlPlacement 296
TRichViewEdit 296
AdjustControlPlacement2 296
TRichViewEdit 296
Align 470
TRVTabInfo 470
Alignment 449, 580
TCustomRVParaInfo 449
TRVLabelItemInfo 580
AllNumbered 459
TRVListInfo 459
AllowSelection 135
TRichView 135
AllowTransparentDrawing 504
TRVReportHelper 504
Animation in TRichView 56
AnimationMode 135
TRichView 135
AppendFrom 172
TRichView 172
AppendRVFFromStream 173
TRichView 173
ApplyListStyle 297
TRichViewEdit 297
ApplyParaStyle 297
TRichViewEdit 297
ApplyParaStyleConversion 298
TRichViewEdit 298
ApplyParaStyleTemplate 298
TRichViewEdit 298
ApplyStyleConversion 299
TRichViewEdit 299
ApplyStyleTemplate 299
TRichViewEdit 299
ApplyTextStyle 300
TRichViewEdit 300
ApplyTextStyleTemplate 301
TRichViewEdit 301
ApplyToParaStyle 464
TRVStyleTemplate 464
ApplyToTextStyle 465
TRVStyleTemplate 465
Assign 443, 446, 455, 458
TCustomRVFontInfo 443
TCustomRVParaInfo 455
TFontInfo 446
TParaInfo 458
AssignDocParameters 488
TRVPrint 488
AssignProperties 547
TRVTableItemInfo 547
AssignSoftPageBreaks 174
TCustomRichView 174
AssignSource 488
TRVPrint 488
AssignStyleTemplates 428
TRVStyleTemplateCollection 428
AssignTo 419, 424, 426, 443
TCustomRVFontInfo 443
TFontInfos 419
TParaInfos 424
TRVListInfos 426
AssignToStrings 428
TRVStyleTemplateCollection 428
Author 277
TRVDocParameters 277
AutoDeleteUnusedStyles 378
TDBRichViewEdit 378
AutoDisplay 374, 379
TDBRichView 374
TDBRichViewEdit 379
AutoHideTableGridLines 269
TRVRTFReaderProperties 269

- B -

BackColor 433
TCustomRVFontInfo 433
Background 450, 594
TCustomRVParaInfo 450
TRVBoxProperties 594
BackgroundBitmap 136
TRichView 136
BackgroundImage 530, 567
TRVTableCellData 567
TRVTableItemInfo 530
BackgroundImageFileName 531, 568
TRVTableCellData 568
TRVTableItemInfo 531
BackgroundStyle 136, 531, 568
TRichView 136
TRVTableCellData 568
TRVTableItemInfo 531
BasePathLinks 269
TRVRTFReaderProperties 269

BeginItemModify 301
TRichViewEdit 301
BeginOleDrag 174
TRichView 174
BeginUndoCustomGroup 302
TRichViewEdit 302
BeginUndoGroup 302
TRichViewEdit 302
BeginUpdate 302
TRichViewEdit 302
BestHeight 568
TRVTableCellData 568
BestWidth 531, 569
TRVTableCellData 569
TRVTableItemInfo 531
BiDiMode 69, 137, 433, 450
TCustomRVFontInfo 433
TCustomRVParaInfo 450
TRichView 137
Bidirectional text in TRichView 69
Border 451, 594
TCustomRVParaInfo 451
TRVBoxProperties 594
BorderColor 532, 569
TRVTableCellData 569
TRVTableItemInfo 532
BorderHSpacing 532
TRVTableItemInfo 532
BorderLightColor 532, 570
TRVTableCellData 570
TRVTableItemInfo 532
BorderOffsets 466, 467
TRVBackgroundRect 466
TRVBorder 467
Borders and background of paragraphs in TRichView 39
BorderStyle 509, 533
TRVScroller 509
TRVTableItemInfo 533
BorderVSpacing 533
TRVTableItemInfo 533
BorderWidth 534
TRVTableItemInfo 534
BottomMargin 137, 277
TRichView 137
TRVDocParameters 277
BoxPosition 592
TRVSidenoteItemInfo 592
BoxProperties 592
TRVSidenoteItemInfo 592
Breaks 93
Building TRichView document 43
Bullets 96
ButtonType 369
TRVSmartPopupProperties 369

- C -

CanChange 302
TRichViewEdit 302
CanMergeCells 547
TRVTableItemInfo 547
CanMergeSelectedCells 548
TRVTableItemInfo 548
CanPaste 303
TRichViewEdit 303
CanPasteRTF 303
TRichViewEdit 303
CanPasteRVF 303
TRichViewEdit 303
Cell Merging 118
CellBorderColor 534
TRVTableItemInfo 534
CellBorderLightColor 534
TRVTableItemInfo 534
CellBorderStyle 535
TRVTableItemInfo 535
CellBorderWidth 535
TRVTableItemInfo 535
CellHPadding 536
TRVTableItemInfo 536
CellHSpacing 536
TRVTableItemInfo 536
CellPadding 536
TRVTableItemInfo 536
Cells 537
TRVTableItemInfo 537
CellVPadding 537
TRVTableItemInfo 537
CellVSpacing 537
TRVTableItemInfo 537
CFRV_HTML 651
CFRV_RTF 651
CFRV_RVF 651
CFRV_URL 651
Change 303, 381
TDBRichViewEdit 381
TRichViewEdit 303
Changed 548
TRVTableItemInfo 548

ChangeListLevels 304
TRichViewEdit 304
ChangeStyleTemplates 304
TRichViewEdit 304
CharScale 434
TCustomRVFontInfo 434
Charset 434
TCustomRVFontInfo 434
CharsetForUnicode 269
TRVRTFReaderProperties 269
CharSpacing 435
TCustomRVFontInfo 435
CheckpointColor 387
TRVStyle 387
CheckpointEvColor 387
TRVStyle 387
Checkpoints in TRichView 30
Clear 175, 304, 517
TCustomRVPrint 517
TRichView 175
TRichViewEdit 304
ClearLeft 138
TRichView 138
ClearLiveSpellingResults 175
TRichView 175
ClearParaFormat 429
TRVStyleTemplateCollection 429
ClearRight 138
TRichView 138
ClearSoftPageBreaks 176
TRichView 176
ClearTextFlow 305
TRichViewEdit 305
ClearTextFormat 429
TRVStyleTemplateCollection 429
ClearUndo 305
TRichViewEdit 305
ClickMode 521
TCustomRVPrintPreview 521
ClientToDocument 176
TRichView 176
ClipMargins 484
TRVPrint 484
ColBandSize 538
TRVTableItemInfo 538
ColCount 538
TRVTableItemInfo 538
Color 139, 369, 387, 435, 467, 468, 521, 538, 570
TCustomRVFontInfo 435
TCustomRVPrintPreview 521
TRichView 139
TRVBackgroundRect 467
TRVBorder 468
TRVSmartPopupProperties 369
TRVStyle 387
TRVTableCellData 570
TRVTableItemInfo 538
ColorMode 514
TCustomRVPrint 514
Colors and layout of tables 114
ColSpan 570
TRVTableCellData 570
Comments 278
TRVDocParameters 278
Component Editor for TRichView 134
ContinuousPrint 488
TRVPrint 488
Controls 94
Controls - Documents - Items 72
ConvertDocToDifferentUnits 176
TRichView 176
ConvertDocToPixels 176
TRichView 176
ConvertDocToTwips 176
TRichView 176
ConvertHighlight 270
TRVRTFReaderProperties 270
ConvertToDifferentUnits 405
TRVStyle 405
ConvertToHotPicture 305
TRichViewEdit 305
ConvertToPicture 305
TRichViewEdit 305
ConvertToPixels 405
TRVStyle 405
ConvertToTwips 405
TRVStyle 405
ConvertToUnit 281
TRVDocParameters 281
ConvertToUnits 489
TRVPrint 489
Copy 177
TRichView 177
CopyDef 177
TRichView 177
CopyImage 177
TRichView 177
CopyRTF 178
TRichView 178
CopyRVF 178

CopyRVF 178
TRichView 178
CopyText 178
TRichView 178
CopyTextA 178
TRichView 178
CopyTextW 178
TRichView 178
Count 503
TRVOfficeCnvList 503
CPEventKind 139
TRichView 139
Create 179, 306, 375, 381, 405, 420, 424, 444, 447, 455, 458, 489, 500, 513, 525, 548, 582, 585, 588, 590, 592, 602, 604, 606, 607
TCustomRVFontInfo 444
TCustomRVParaInfo 455
TCustomRVPrintPreview 525
TDBRichView 375
TDBRichViewEdit 381
TFontInfo 447
TFontInfos 420
TParaInfo 458
TParaInfos 424
TRichView 179
TRichViewEdit 306
TRVEndnoteItemInfo 588
TRVFootnoteItemInfo 590
TRVLabelItemInfo 582
TRVNoteReferenceItemInfo 604
TRVOfficeConverter 500
TRVPageCountItemInfo 607
TRVPageNumberItemInfo 606
TRVPrint 489
TRVScroller 513
TRVSeqItemInfo 585
TRVSidenoteItemInfo 592
TRVStyle 405
TRVTableItemInfo 548
TRVTextBoxItemInfo 602
CreateEx 549, 582, 585, 587, 589, 591, 593, 602, 604, 606, 607
TCustomRVNoteItemInfo 587
TRVEndnoteItemInfo 589
TRVFootnoteItemInfo 591
TRVLabelItemInfo 582
TRVNoteReferenceItemInfo 604
TRVPageCountItemInfo 607
TRVPageNumberItemInfo 606
TRVSeqItemInfo 585
TRVSidenoteItemInfo 593
TRVTableItemInfo 549
TRVTextBoxItemInfo 602
crJump 77
crRVZoomIn 77
crRVZoomOut 77
CurItemNo 288
TRichViewEdit 288
CurItemStyle 289
TRichViewEdit 289
CurParaStyleNo 289
TRichViewEdit 289
CurrentItemColor 388
TRVStyle 388
Cursor 140, 580
TRichView 140
TRVLabelItemInfo 580
Cursors in TRichView 77
CurTextStyleNo 290
TRichViewEdit 290
Custom drawing in TRichView 76
Custom item types in TRichView 111
CustomCaretInterval 290
TRichViewEdit 290
CutDef 306
TRichViewEdit 306

- D -

DataField 375, 379
TDBRichView 375
TDBRichViewEdit 379
DataSource 375, 379
TDBRichView 375
TDBRichViewEdit 379
DefaultPictureVAlign 291
TRichViewEdit 291
DefCodePage 388
TRVStyle 388
DefStyleName 427
TRVStyleTemplateCollection 427
DefStyleNo 457
TParaInfo 457
DefTabWidth 388
TRVStyle 388
DefUnicodeStyle 389
TRVStyle 389
DeleteCols 549
TRVTableItemInfo 549
DeleteEmptyCols 550
TRVTableItemInfo 550

DeleteEmptyRows 550
TRVTableItemInfo 550
DeleteItems 179
TRichView 179
DeleteList 469
TRVTabInfos 469
DeleteMarkedStyles 180
TRichView 180
DeleteParas 180
TRichView 180
DeleteRows 550
TRVTableItemInfo 550
DeleteSection 180
TRichView 180
DeleteSelectedCols 551
TRVTableItemInfo 551
DeleteSelectedRows 551
TRVTableItemInfo 551
DeleteSelection 306
TRichViewEdit 306
DeleteTextStyle 405
TRVStyle 405
DeleteUnusedStyles 181
TRichView 181
Delimiters 140
TRichView 140
Deselect 181, 551
TRichView 181
TRVTableItemInfo 551
Destroy 181, 306, 376, 381, 405, 500, 551
TDBRichView 376
TDBRichViewEdit 381
TRichView 181
TRichViewEdit 306
TRVOfficeConverter 500
TRVStyle 405
TRVTableItemInfo 551
DifferentUnitsToUnits 406
TRVStyle 406
DisabledFontColor 389
TRVStyle 389
DocParameters 140
TRichView 140
DocProperties 141
TRichView 141
Document 586
TCustomRVNoteItemInfo 586
DocumentHeight 142
TRichView 142
DocumentToClient 176
TRichView 176
DoInPaletteMode 142
TRichView 142
Drag and drop in TRichView 55
Drag&Drop in TRichView 55
Draw 447
TFontInfo 447
DrawButton 616
DrawControl 616
DrawEdit 616
DrawMemo 616
DrawPage 505
TRVReportHelper 505
DrawPageAt 506
TRVReportHelper 506
DrawPanel 616
DrawPreview 489
TRVPrint 489

- E -

Edit 610
TCustomRVData 610
EditCell 552
TRVTableItemInfo 552
Editing Cells 117
EditorOptions 291
TRichViewEdit 291
EndAt 514
TCustomRVPrint 514
EndItemModify 307
TRichViewEdit 307
EndnoteNumbering 389
TRVStyle 389
Endnotes 103
EndUpdate 307
TRichViewEdit 307
ERichViewError 608
ErrorCode 270, 497
TRVOfficeConverter 497
TRVRTFReaderProperties 270
EvenColumnsColor 546
TRVTableItemInfo 546
EvenRowsColor 546
TRVTableItemInfo 546
Example
Converting table to text 125
Dividing table 126
Inserting controls what can modify themselves in DBRichViewEdit 383

Example
Loading UTF-8 files 282
Moving caret to the beginning of paragraph 371
OnDropFiles example 372
Using viewer-style methods in DBRichViewEdit 383
ExcludeDocXExportConverter 498
TRVOfficeConverter 498
ExcludeDocXImportConverter 498
TRVOfficeConverter 498
ExcludeHTMLExportConverter 498
TRVOfficeConverter 498
ExcludeHTMLImportConverter 498
TRVOfficeConverter 498
Export of Tables 123
Export of TRichView contents to HTML 62
ExportConverters 499
TRVOfficeConverter 499
ExportRTF 500
TRVOfficeConverter 500
ExportRV 500
TRVOfficeConverter 500
ExtensionsInFilter 499
TRVOfficeConverter 499
ExtractMetafileBitmaps 271
TRVRTFReaderProperties 271

- F -

FacingPages 278, 484
TRVDocParameters 278
TRVPrint 484
Field 375, 379
TDBRichView 375
TDBRichViewEdit 379
FieldFormat 380
TDBRichViewEdit 380
FieldHighlightColor 390
TRVStyle 390
FieldHighlightType 390
TRVStyle 390
Filter 503
TRVOfficeConverterInfo 503
Find 469
TRVTabInfos 469
FindById 429
TRVStyleTemplateCollection 429
FindByName 429
TRVStyleTemplateCollection 429
FindCheckpointByName 182
TRichView 182
FindCheckpointByTag 182
TRichView 182
FindControlItemNo 182
TRichView 182
FindItemById 429
TRVStyleTemplateCollection 429
FindItemByName 429
TRVStyleTemplateCollection 429
FindParaStyle 407
TRVStyle 407
FindStyleWithAlignment 424
TParaInfos 424
FindStyleWithCharset 420
TFontInfos 420
FindStyleWithColor 420
TFontInfos 420
FindStyleWithFont 420
TFontInfos 420
FindStyleWithFontName 421
TFontInfos 421
FindStyleWithFontSize 421
TFontInfos 421
FindStyleWithFontStyle 421
TFontInfos 421
FindStyleWithLevels 426
TRVListInfos 426
FindSuchStyle 422, 425, 426
TFontInfos 422
TParaInfos 425
TRVListInfos 426
FindSuchStyleEx 422, 425
TFontInfos 422
TParaInfos 425
FindTextStyle 407
TRVStyle 407
Finished 506
TRVReportHelper 506
First 525
TCustomRVPrintPreview 525
FirstColumnColor 546
TRVTableItemInfo 546
FirstIndent 451, 473
TCustomRVParaInfo 451
TRVListLevel 473
FirstItemVisible 144
TRichView 144
FirstJumpNo 144
TRichView 144
FirstPageNo 514

FirstPageNo 514
TCustomRVPrint 514
FixMarginsMode 484
TRVPrint 484
FloatingLineColor 390
TRVStyle 390
Font 473
TRVListLevel 473
FontName 436
TCustomRVFontInfo 436
FontQuality 391
TRVStyle 391
FooterY 278, 485
TRVDocParameters 278
TRVPrint 485
FootnoteNumbering 389
TRVStyle 389
FootnotePageReset 392
TRVStyle 392
Footnotes 104
ForbiddenIds 428
TRVStyleTemplateCollection 428
ForceFieldHighlight 293
TRichViewEdit 293
Format 182
TRichView 182
FormatNextPage 506
TRVReportHelper 506
FormatPages 490
TRVPrint 490
FormatString 474, 584
TRVListLevel 474
TRVSeqItemInfo 584
FormatStringW 475
TRVListLevel 475
FormatTail 183
TRichView 183
FullRedraw 392
TRVStyle 392

- G -

GetAllText 620
GetAsDifferentUnits 406
TRVStyle 406
GetAsPixels 406
TRVStyle 406
GetAsRVUnits 406
TRVStyle 406
GetAsTwips 406
TRVStyle 406
GetBreakInfo 183
TRichView 183
GetBulletInfo 184
TRichView 184
GetCellAt 552
TRVTableItemInfo 552
GetCheckpointAtCaret 307
TRichViewEdit 307
GetCheckpointByNo 185
TRichView 185
GetCheckpointInfo 185
TRichView 185
GetCheckpointItemNo 186
TRichView 186
GetCheckpointNo 186
TRichView 186
GetCheckpointXY 186
TRichView 186
GetCheckpointY 186
TRichView 186
GetCheckpointYEx 187
TRichView 187
GetControlInfo 187
TRichView 187
GetCurrentBreakInfo 308
TRichViewEdit 308
GetCurrentBulletInfo 308
TRichViewEdit 308
GetCurrentChar 620
GetCurrentCheckpoint 309
TRichViewEdit 309
GetCurrentControlInfo 309
TRichViewEdit 309
GetCurrentHotspotInfo 310
TRichViewEdit 310
GetCurrentItem 311
TRichViewEdit 311
GetCurrentItemEx 311
TRichViewEdit 311
GetCurrentItemExtraIntProperty 312
GetCurrentItemExtraIntPropertyEx 312
TRichViewEdit 312
GetCurrentItemExtraStrProperty 313
TRichViewEdit 313
GetCurrentItemExtraStrPropertyEx 313
TRichViewEdit 313
GetCurrentItemText 313
TRichViewEdit 313
GetCurrentItemTextA 313

GetCurrentItemTextA 313
TRichViewEdit 313
GetCurrentItemTextW 313
TRichViewEdit 313
GetCurrentItemVAlign 314
TRichViewEdit 314
GetCurrentLineCol 314
TRichViewEdit 314
GetCurrentLineText 620
GetCurrentMisspelling 315
TRichViewEdit 315
GetCurrentParaSectionText 620
GetCurrentParaText 620
GetCurrentPictureInfo 315
TRichViewEdit 315
GetCurrentTag 316
TRichViewEdit 316
GetCurrentTextInfo 316
TRichViewEdit 316
GetCurrentWord 620
GetEditedCell 552
TRVTableItemInfo 552
GetExportFilter 501
TRVOfficeConverter 501
GetFirstCheckpoint 188
TRichView 188
GetFirstItemOnPage 517
TCustomRVPrint 517
GetFocusedItem 188
TRichView 188
GetFooterRect 490
TRVPrint 490
GetHeaderRect 490
TRVPrint 490
GetHotspotInfo 189
TRichView 189
GetImportFilter 501
TRVOfficeConverter 501
GetItem 190
TRichView 190
GetItemAt 190
TCustomRichView 190
GetItemCheckpoint 191
TRichView 191
GetItemClientCoords 191
TRichView 191
GetItemCoords 191
TRichView 191
GetItemCoordsEx 192
TRichView 192
GetItemExtraIntProperty 193
TRichView 193
GetItemExtraIntPropertyEx 193
TRichView 193
GetItemExtraStrProperty 193
TCustomRichView 193
GetItemExtraStrPropertyEx 193
TCustomRichView 193
GetItemNo 194
TRichView 194
GetItemPara 194
TRichView 194
GetItemStyle 194
TRichView 194
GetItemTag 195
TRichView 195
GetItemText 195
TRichView 195
GetItemTextA 195
TRichView 195
GetItemTextW 195
TRichView 195
GetItemVAlign 196
TRichView 196
GetJumpPointItemNo 196
TRichView 196
GetJumpPointLocation 197
TRichView 197
GetJumpPointY 197
TRichView 197
GetLastCheckpoint 198
TRichView 198
GetLastPageHeight 506
TRVReportHelper 506
GetLineNo 198
TRichView 198
GetListMarkerInfo 199
TRichView 199
GetListStyleClass 406
TRVStyle 406
GetMainCell 579
TRVTableRows 579
GetNextCheckpoint 199
TRichView 199
GetNextFileName 241
GetNormalizedSelectionBounds 553
TRVTableItemInfo 553
GetOffsAfterItem 200
TRichView 200
GetOffsBeforeItem 201

GetOffsBeforeItem 201
TRichView 201
GetOriginEx 611
TCustomRVFormattedData 611
GetPageNo 518
TCustomRVPrint 518
GetParaStyleClass 407
TRVStyle 407
GetPictureInfo 201
TRichView 201
GetPrevCheckpoint 202
TRichView 202
GetRVData 575, 611
TCustomRVData 611
TRVTableCellData 575
GetRVDataText 620
GetRVESearchOptions 616
GetRVSearchOptions 616
GetSelectedImage 202
TRichView 202
GetSelectionBounds 203, 553
TRichView 203
TRVTableItemInfo 553
GetSelText 204
TRichView 204
GetSelTextA 204
TRichView 204
GetSelTextW 204
TRichView 204
GetSourceRVData 611
TCustomRVData 611
GetTextInfo 204
TRichView 204
GetTextStyleClass 407
TRVStyle 407
GetVisibleText 620
GetWordAt 205
TRichView 205
GetWordAtA 205
TRichView 205
GetWordAtW 205
TRichView 205
GridColor 392
TRVStyle 392
GridReadOnlyColor 392
TRVStyle 392
GridReadOnlyStyle 392
TRVStyle 392
GridStyle 392
TRVStyle 392

- H -

HasNumbering 460, 479
TRVListInfo 460
TRVListLevel 479
HeaderY 278, 485
TRVDocParameters 278
TRVPrint 485
HeadingRowColor 546
TRVTableItemInfo 546
HeadingRowCount 539
TRVTableItemInfo 539
Height 594
TRVBoxProperties 594
HeightType 594
TRVBoxProperties 594
Hidden text in TRichView 42
Hint 370, 571
TRVSmartPopupProperties 370
TRVTableCellData 571
HorizontalAlignment 597
TRVBoxPosition 597
HorizontalAnchor 596
TRVBoxPosition 596
HorizontalOffset 597
TRVBoxPosition 597
HorizontalPositionKind 597
TRVBoxPosition 597
Hot-Pictures 93
Hotspots 97
HOutermostRule 539
TRVTableItemInfo 539
HoverBackColor 436
TCustomRVFontInfo 436
HoverColor 370, 393, 436
TCustomRVFontInfo 436
TRVSmartPopupProperties 370
TRVStyle 393
HoverEffects 437
TCustomRVFontInfo 437
HoverLineColor 370
TRVSmartPopupProperties 370
HoverUnderlineColor 437
TCustomRVFontInfo 437
How to
(List) 665
change page size and orientation for printing
RichView 666

How to
combine several TRichView documents in one 671
copy one TRichView to another 172
create a chat window 673
implement commands like "make bold", "apply font" 667
implement Find and Replace dialogs 667
implement headers and footers 666
implement smart indents 671
make a plain text editor 669
make Unicode editor 669
move a caret to the beginning or to the end of document in TRichViewEdit 668
remove text formatting 673
switch insert/overtype mode 668
use third-party graphic classes with RichView 671
HRuleColor 539
TRVTableItemInfo 539
HRuleWidth 540
TRVTableItemInfo 540
HScrollMax 509
TRVScroller 509
HScrollPos 510
TRVScroller 510
HScrollVisible 510
TRVScroller 510
HTML and TRichView 62
HTMLFormatName 651
Hypertext in TRichView 35

- I -

Id 461
TRVStyleTemplate 461
IgnoreBookmarks 271
TRVRTFReaderProperties 271
IgnoreEscape 380
TDBRichViewEdit 380
IgnoreFields 271
TRVRTFReaderProperties 271
IgnoreNotes 271
TRVRTFReaderProperties 271
IgnorePageBreaks 515
TCustomRVPrint 515
IgnorePictures 271
TRVRTFReaderProperties 271
IgnoreSequences 272
TRVRTFReaderProperties 272
IgnoreTables 272
TRVRTFReaderProperties 272
ImageIndex 370, 475
TRVListLevel 475
TRVSmartPopupProperties 370
ImageList 370, 475
TRVListLevel 475
TRVSmartPopupProperties 370
imgSaveNo 241
ImportConverters 499
TRVOfficeConverter 499
ImportRTF 501
TRVOfficeConverter 501
ImportRV 502
TRVOfficeConverter 502
InactiveSelColor 393
TRVStyle 393
InactiveSelTextColor 393
TRVStyle 393
Init 507
TRVReportHelper 507
InplaceEditor 510
TRVScroller 510
InsertBreak 317
TRichViewEdit 317
InsertBullet 317
TRichViewEdit 317
InsertCheckpoint 318
TRichViewEdit 318
InsertCols 554
TRVTableItemInfo 554
InsertColsLeft 554
TRVTableItemInfo 554
InsertColsRight 554
TRVTableItemInfo 554
InsertControl 318
TRichViewEdit 318
InsertHotPicture 319
TRichViewEdit 319
InsertHotspot 319
TRichViewEdit 319
InsertItem 320
TRichViewEdit 320
InsertOEMTextFromFile 321
TRichViewEdit 321
InsertPageBreak 321
TRichViewEdit 321
InsertPicture 322
TRichViewEdit 322
InsertRows 555
TRVTableItemInfo 555

InsertRowsAbove 555
TRVTableItemInfo 555
InsertRowsBelow 555
TRVTableItemInfo 555
InsertRTFFromFileEd 323
TRichViewEdit 323
InsertRTFFromStreamEd 323
TRichViewEdit 323
InsertRVFFromFileEd 324
TRichViewEdit 324
InsertRVFFromStream 206
TRichView 206
InsertRVFFromStreamEd 325
TRichViewEdit 325
InsertStringATag 326
TRichViewEdit 326
InsertStringTag 326
TRichViewEdit 326
InsertStringWTag 326
TRichViewEdit 326
InsertTab 326
TRichViewEdit 326
InsertText 327
TRichViewEdit 327
InsertTextA 327
TRichViewEdit 327
InsertTextFromFile 328
TRichViewEdit 328
InsertTextFromFileW 328
TRichViewEdit 328
InsertTextW 327
TRichViewEdit 327
InternalWidth 468
TRVBorder 468
Intersect 470
TRVTabInfos 470
InvalidItem 418, 423
TFontInfos 418
TParaInfos 423
InvalidPicture 393
TRVStyle 393
IsCellSelected 556
TRVTableItemInfo 556
IsEqual 444, 448, 456, 458
TCustomRVFontInfo 444
TCustomRVParaInfo 456
TFontInfo 448
TParaInfo 458
IsFromNewLine 207
TRichView 207
IsParaStart 208
TRichView 208
IsValidImporter 502
TRVOfficeConverter 502
Item types
Breaks 93
Bullets 96
Controls 94
Endnotes 103
Footnotes 104
Hot-Pictures 93
Hotspots 97
Labels 101
List Markers 99
Numbered sequences 102
Page counts 110
Page numbers 109
Pictures 90
References to parent endnotes and footnotes 108
Sidenotes 105
Tables 98
Tabulators 89
Text 88
Text boxes 107
Item types in TRichView 85, 526
ItemCount 144
TRichView 144
Items 418, 423, 426, 428, 469, 472, 503, 577, 579
TFontInfos 418
TParaInfos 423
TRVListInfos 426
TRVListLevelCollection 472
TRVOfficeCnvList 503
TRVStyleTemplateCollection 428
TRVTabInfos 469
TRVTableRow 577
TRVTableRows 579

- J -

Jump 445
TFontInfo 445
JumpCursor 394, 438
TCustomRVFontInfo 438
TRVStyle 394

- K -

KeepTogether 577
TRVTableRow 577

Kind 461
TRVStyleTemplate 461

- L -

Labels 101
Last 525
TCustomRVPrintPreview 525
LastColumnColor 546
TRVTableItemInfo 546
LastItemVisible 144
TRichView 144
LastLineAlignment 449
TCustomRVParaInfo 449
LastPageNo 515
TCustomRVPrint 515
LastRowColor 546
TRVTableItemInfo 546
Leader 470
TRVTabInfo 470
LeftIndent 451, 476
TCustomRVParaInfo 451
TRVListLevel 476
LeftMargin 145, 278
TRichView 145
TRVDocParameters 278
Levels 459
TRVListInfo 459
LinearToRichView 621
LineBreaksAsParagraphs 272
TRVRTFReaderProperties 272
LineColor 370
TRVSmartPopupProperties 370
LineCount 145
TRichView 145
LineSelectCursor 394
TRVStyle 394
LineSpacing 452
TCustomRVParaInfo 452
LineSpacingType 452
TCustomRVParaInfo 452
LineWrapMode 394
TRVStyle 394
List Markers 99
ListStyles 396
TRVStyle 396
ListType 476
TRVListLevel 476
Live spelling check in TRichView 69
LiveSpellingColor 397
TRVStyle 397
LiveSpellingMode 293
TRichViewEdit 293
LiveSpellingValidateWord 208
TRichView 208
LoadField 376, 381
TDBRichView 376
TDBRichViewEdit 381
LoadFromRVST 429
TRVStyleTemplateCollection 429
LoadFromStream 208, 556
TRichView 208
TRVTableItemInfo 556
LoadINI 407
TRVStyle 407
LoadReg 408
TRVStyle 408
LoadRTF 209
TRichView 209
LoadRTFFromStream 210
TRichView 210
LoadRVF 211
TRichView 211
LoadRVFFromStream 211
TRichView 211
LoadText 212
TRichView 212
LoadTextFromStream 213
TRichView 213
LoadTextFromStreamW 213
TRichView 213
LoadTextW 212
TRichView 212

- M -

MainRVStyle 397
TRVStyle 397
MakePreview 491
TRVPrint 491
MakeScaledPreview 491
TRVPrint 491
Margins 486
TRVPrint 486
MarginsPen 522
TCustomRVPrintPreview 522
MarkerAlignment 477
TRVListLevel 477
MarkerIndent 478
TRVListLevel 478

MarkStylesInUse 213
TRichView 213
MaxLength 145
TRichView 145
MaxPrintedItemNo 515
TCustomRVPrint 515
MaxTextWidth 146
TRichView 146
Menu 371
TRVSmartPopupProperties 371
MergeCells 556
TRVTableItemInfo 556
MergeSelectedCells 557
TRVTableItemInfo 557
MetafileCompatibility 515
TCustomRVPrint 515
MinPrintedItemNo 516
TCustomRVPrint 516
MinTextWidth 146
TRichView 146
MinWidth 580
TRVLabelItemInfo 580
MirrorMargins 279, 487
TRVDocParameters 279
TRVPrint 487
Modified 293
TRichViewEdit 293
ModifiedProperties 445, 457
TFontInfo 445
TParaInfo 457
Modifying TRichView items 51

- N -

Name 461, 503
TRVOfficeConverterInfo 503
TRVStyleTemplate 461
Next 462, 526
TCustomRVPrintPreview 526
TRVStyleTemplate 462
NextId 462
TRVStyleTemplate 462
NextParaNo 457
TParaInfo 457
NextStyleNo 446
TFontInfo 446
NoMetafiles 505
TRVReportHelper 505
NormalStyleTemplate 428
TRVStyleTemplateCollection 428
NoteText 146, 587
TCustomRVNoteItemInfo 587
TRichView 146
NoVScroll 511
TRVScroller 511
Numbered sequences 102
NumberType 584, 605
TCustomRVPageNumberItemInfo 605
TRVSeqItemInfo 584

- O -

Obtaining items of TRichView 50
OddColumnsColor 546
TRVTableItemInfo 546
OddRowsColor 546
TRVTableItemInfo 546
OffsetInCurItem 294
TRichViewEdit 294
OnAddStyle 238
TRichView 238
OnAfterApplyStyle 410
TRVStyle 410
OnApplyStyle 410
TRVStyle 410
OnApplyStyleColor 411
TRVStyle 411
OnCaretGetOut 356
TRichViewEdit 356
OnCaretMove 356
TRichViewEdit 356
OnCellEditing 564
TRVTableItemInfo 564
OnChange 357
TRichViewEdit 357
OnChanging 357
TRichViewEdit 357
OnCheckpointVisible 238
TRichView 238
OnCheckStickingItems 357
TRichViewEdit 357
OnControlAction 239
TRichView 239
OnConverting 503
TRVOfficeConverter 503
OnCopy 240
TRichView 240
OnCurParaStyleChanged 358
TRichViewEdit 358
OnCurTextStyleChanged 358

OnCurTextStyleChanged 358
TRichViewEdit 358
OnDrawBorder 565
TRVTableItemInfo 565
OnDrawCheckpoint 411, 507
TRVReportHelper 507
TRVStyle 411
OnDrawCustomCaret 358
TRichViewEdit 358
OnDrawHyperlink 508
TRVReportHelper 508
OnDrawPageBreak 413
TRVStyle 413
OnDrawParaBack 414
TRVStyle 414
OnDrawStyleText 415
TRVStyle 415
OnDrawTextBack 416
TRVStyle 416
OnDropFiles 359
TRichViewEdit 359
OneLevelPreview 459
TRVListInfo 459
OnFormatting 494
TRVPrint 494
OnGetItemCursor 240
TRichView 240
OnGetPageNumber 508
TRVReportHelper 508
OnHScrolled 513
TRVScroller 513
OnHTMLSaveImage 241
TRichView 241
OnImportPicture 242
TRichView 242
OnItemAction 244
TRichView 244
OnItemHint 246
TRichView 246
OnItemResize 359
TRichViewEdit 359
OnItemTextEdit 360
TRichViewEdit 360
OnJump 247
TRichView 247
OnLoadCustomFormat 376, 381
TDBRichView 376
TDBRichViewEdit 381
OnLoadDocument 376, 382
TDBRichView 376
TDBRichViewEdit 382
OnMeasureCustomCaret 360
TRichViewEdit 360
OnNewDocument 377, 382
TDBRichView 377
TDBRichViewEdit 382
OnOleDragEnter 361
TRichViewEdit 361
OnOleDragLeave 362
TRichViewEdit 362
OnOleDragOver 362
TRichViewEdit 362
OnOleDrop 363
TRichViewEdit 363
OnPagePostpaint 495
TRVPrint 495
OnPagePrepaint 495
TRVPrint 495
OnPaint 247
TRichView 247
OnParaStyleConversion 364
TRichViewEdit 364
OnPaste 365
TRichViewEdit 365
OnPrintComponent 519
TCustomRVPrint 519
OnProgress 248
TRichView 248
OnReadHyperlink 248
TCustomRichView 248
OnRVDblClick 250
TRichView 250
OnRVFControlNeeded 251
TRichView 251
OnRVFImageListNeeded 252
TRichView 252
OnRVFPictureNeeded 252
TRichView 252
OnRVMouseDown 253
TRichView 253
OnRVMouseMove 254
TRichView 254
OnRVMouseUp 254
TRichView 254
OnRVRightClick 255
TRichView 255
OnSaveComponentToFile 256
TRichView 256
OnSaveCustomFormat 382
TDBRichViewEdit 382

OnSaveDocXExtra 257
TRichView 257
OnSaveHTMLExtra 259
TRichView 259
OnSaveImage2 260
TRichView 260
OnSaveItemToFile 261
TRichView 261
OnSaveParaToHTML 263
TRichView 263
OnSaveRTFExtra 263
TRichView 263
OnSelect 265
TRichView 265
OnSendingToPrinter 496
TRVPrint 496
OnSmartPopupClick 366
TRichViewEdit 366
OnSpellingCheck 265
TRichView 265
OnStyleConversion 366
TRichViewEdit 366
OnStyleHoverSensitive 417
TRVStyle 417
OnStyleTemplatesChange 266
TRichView 266
OnURLNeeded 266
TRichView 266
OnVScrolled 513
TRVScroller 513
OnWriteHyperlink 267
TRichView 267
OnZoomChanged 526
TCustomRVPrintPreview 526
Opacity 467, 540, 571
TRVBackgroundRect 467
TRVTableCellData 571
TRVTableItemInfo 540
Options 146, 438, 453, 478, 540, 571
TCustomRVFontInfo 438
TCustomRVParaInfo 453
TRichView 146
TRVListLevel 478
TRVTableCellData 571
TRVTableItemInfo 540
Orientation 279
TRVDocParameters 279
OutlineLevel 454
TCustomRVParaInfo 454
Overview 28
Overview of item types 85, 86
Overview of tables 111

- P -

Page counts 110
Page numbers 109
PageBorderColor 522
TCustomRVPrintPreview 522
PageBorderWidth 522
TCustomRVPrintPreview 522
PageBreakBefore 578
TRVTableRow 578
PageBreakColor 398
TRVStyle 398
PageBreaksBeforeItems 149
TRichView 149
PageHeight 279
TRVDocParameters 279
PageNo 522
TCustomRVPrintPreview 522
PageNoFromNumber 516
TCustomRVPrint 516
PagesCount 516
TCustomRVPrint 516
PageWidth 279
TRVDocParameters 279
Paragraphs in TRichView 37
ParaStyle 463
TRVStyleTemplate 463
ParaStyleMode 272
TRVRTFReaderProperties 272
ParaStyleNo 273
TRVRTFReaderProperties 273
ParaStyles 398
TRVStyle 398
ParaStyleTemplateId 439
TCustomRVFontInfo 439
Parent 463
TRVStyleTemplate 463
ParentId 463
TRVStyleTemplate 463
Paste 328
TRichViewEdit 328
PasteBitmap 329
TRichViewEdit 329
PasteGraphicFile 329
TRichViewEdit 329
PasteMetafile 330
TRichViewEdit 330

PasteRTF 330
TRichViewEdit 330
PasteRVF 331
TRichViewEdit 331
PasteText 331
TRichViewEdit 331
PasteTextA 331
TRichViewEdit 331
PasteTextW 331
TRichViewEdit 331
PasteURL 332
TRichViewEdit 332
Path 503
TRVOfficeConverterInfo 503
Picture 479
TRVListLevel 479
Pictures 90
PixelsPerInch 419
TFontInfos 419
PixelsToUnits 406
TRVStyle 406
Position 371, 471
TRVSmartPopupProperties 371
TRVTabInfo 471
PositionInText 599
TRVBoxPosition 599
Prev 526
TCustomRVPrintPreview 526
Preview100PercentHeight 487
TRVPrint 487
Preview100PercentWidth 487
TRVPrint 487
PreviewCorrection 516
TCustomRVPrint 516
PreviewMode 499
TRVOfficeConverter 499
Print 492
TRVPrint 492
PrintableAreaPen 523
TCustomRVPrintPreview 523
PrintOptions 542
TRVTableItemInfo 542
PrintPages 492
TRVPrint 492
Protection 439
TCustomRVFontInfo 439
ProtectTextStyleNo 581
TRVLabelItemInfo 581

- Q -

QuickAccess 463
TRVStyleTemplate 463

- R -

ReadDocParameters 273
TRVRTFReaderProperties 273
ReadOnly 294, 380
TDBRichViewEdit 380
TRichViewEdit 294
Redo 332
TRichViewEdit 332
RedoAction 333
TRichViewEdit 333
RedoName 333
TRichViewEdit 333
References to parent endnotes and footnotes 108
Reformat 214
TRichView 214
RefreshListMarkers 214
TRichView 214
RefreshSequences 214
TRichView 214
RelativeToCell 599
TRVBoxPosition 599
RemoveCheckpoint 214
TRichView 214
RemoveCheckpointAtCaret 333
TRichViewEdit 333
RemoveCheckpointEd 333
TRichViewEdit 333
RemoveCurrentCheckpoint 334
TRichViewEdit 334
RemoveCurrentPageBreak 334
TRichViewEdit 334
RemoveInternalLeading 581
TRVLabelItemInfo 581
RemoveLists 334
TRichViewEdit 334
ReplaceDocumentEd 587
TCustomRVNoteItemInfo 587
Reset 281, 584
TRVDocParameters 281
TRVSeqItemInfo 584
ResetAnimation 215
TRichView 215
ResetLayout 281

ResetLayout 281
TRVDocParameters 281
ResetNameCounter 430
TRVStyleTemplateCollection 430
ResizeControl 335
TRichViewEdit 335
ResizeCurrentControl 335
TRichViewEdit 335
RichView 26, 505
TRVReportHelper 505
RichView Format (RVF) overview 59
RichViewAllowCopyTableCells 652
RichViewAlternativePicPrint 652
RichViewApostropheInWord 653
RichViewAutoAssignNormalStyleTemplate 653
RichViewCompareStyleNames 653
RichViewDoNotCheckRVFStyleRefs 653
RichViewDoNotMergeNumbering 654
RichViewEditCaretHeightExtra 652
RichViewEditCaretWidth 652
RichViewEditDefaultProportionalResize 654
RichViewEditEnterAllowsEmptyMarkeredLines 654
RichViewEditMaxCaretHeight 652
RichViewJustifyBeforeLineBreak 654
RichViewMaxThumbnailCount 655
RichViewPixelsPerInch 655
RichViewResetStandardFlag 656
RichViewSaveDivInHTMLEx 652
RichViewSaveHyperlinksInDocXAsFields 656
RichViewSavePageBreaksInText 652
RichViewSavePInHTML 652
RichViewShowGhostSpaces 656
RichViewStringFormatMethod 656
RichViewTableAutoAddRow 657
RichViewTableDefaultRTFAutofit 657
RichViewToLinear 621
RichViewUnicodeInput 657
RightIndent 454
TCustomRVParaInfo 454
RightMargin 149, 279
TRichView 149
TRVDocParameters 279
Rotation 573
TRVTableCellData 573
RowBandSize 538
TRVTableItemInfo 538
RowCount 543
TRVTableItemInfo 543
Rows 543
TRVTableItemInfo 543
RowSpan 574
TRVTableCellData 574
RTFOptions 150
TRichView 150
RTFormatName 651
RTFReadProperties 151
TRichView 151
Rules of tables 114
RV_AfterImportGraphic 671
RV_CreateGraphics 671
RV_GetPrinterDC 617
RV_RegisterHTMLGraphicFormat 671
RV_RegisterPngGraphic 671
RV_TestFileUnicode 620
RV_TestStreamUnicode 620
RV_TestStringUnicode 620
RV_TwipsToUnits 622
RV_UnitsPerInch 622
RV_UnitsToPixels 622
RV_UnitsToTwips 622
RV_UnitsToUnits 622
RVAllFontInfoProperties 633
RVAllParaInfoProperties 637
RVCharsPerLineBreak 621
RVData 151
TRichView 151
rvdoALL 626
rveipcAlignment 658, 661
rveipcCursor 658, 661
rveipcMinWidth 658, 661
rveipcProtectTextStyleNo 658, 661
rveipcRemoveInternalLeading 658, 661
rveipcSeqNumberType 658, 661
rveipcSeqReset 658, 661
rveipcSeqStartFrom 658, 661
RVF 59, 77, 140
RVF Specification 77
RVFOptions 152
TRichView 152
RVFormatName 651
RVFParaStylesReadMode 154
TRichView 154
RVFTextStylesReadMode 154
TRichView 154
RVFWarnings 155
TRichView 155
RVGetFirstEndnote 618
RVGetFirstFootnote 618

RVGetFirstSidenote 618
RVGetLinearCaretPos 621
RVGetNextEndnote 618
RVGetNextFootnote 618
RVGetNextSidenote 618
RVGetNoteTextStyleNo 619
RVGetSelection 621
RVGetSelectionEx 621
RVGetTextLength 621
RVGetTextRange 621
RVGraphicHandler 662
RVIsCustomURL procedural 662
RVIsEmail functions 619
RVIsURL functions 619
RVNonTextCharacter 621
RVParagraphMarks 662
RVPrint 384
  TRVPrintPreview 384
rvsBreak 663
rvsBullet 663
rvsComponent 663
rvsDefStyle 663
rvsEndnote 663
RVSetLinearCaretPos 621
RVSetSelection 621
RVSetSelectionEx 621
rvsFootnote 663
rvsHeading 663
rvsHotPicture 663
rvsHotspot 663
rvsJump1 663
rvsJump2 663
rvsKeyword 663
rvsLabel 663
rvsListMarker 663
rvsNormal 663
rvsNoteReference 663
rvsPicture 663
rvsSequence 663
rvsSubheading 663
rvsTab 663
rvsTable 663
RVStyle.TRVColorMode 625
RVThumbnailMaker 664
RVU_AnsiToUnicode 620
RVU_AnsiToUTF8 620
RVU_GetRawUnicode 620
RVU_RawUnicodeToWideString 620
RVU_UnicodeToAnsi 620
RVUnitsToUnits 406
  TRVStyle 406
RVVisibleSpecialCharacters 665

- S -

SaveCSS 408
  TRVStyle 408
SaveCSSToStream 409
  TRVStyle 409
SaveDocX 215
  TRichView 215
SaveDocXToStream 216
  TRichView 216
SaveHTML 216
  TRichView 216
SaveHTMLEx 217
  TRichView 217
SaveHTMLToStream 218
  TRichView 218
SaveHTMLToStreamEx 218
  TRichView 218
SaveINI 409
  TRVStyle 409
SavePicture 219
  TRichView 219
SaveReg 409
  TRVStyle 409
SaveRowsToStream 557
  TRVTableItemInfo 557
SaveRTF 219
  TRichView 219
SaveRTFToStream 220
  TRichView 220
SaveRVF 220
  TRichView 220
SaveRVFToStream 220
  TRichView 220
SaveText 221
  TRichView 221
SaveTextToStream 221
  TRichView 221
SaveTextToStreamW 221
  TRichView 221
SaveTextW 221
  TRichView 221
SaveToRVST 429
  TRVStyleTemplateCollection 429
SaveToStream 557
  TRVTableItemInfo 557

Scrolling in TRichView 46
ScrollTo 513
TRVScroller 513
Searching and replacing in TRichView and TRichViewEdit 49
SearchText 222, 336
TRichView 222
TRichViewEdit 336
SearchTextA 222, 336
TRichView 222
TRichViewEdit 336
SearchTextW 222, 336
TRichView 222
TRichViewEdit 336
SelColor 393
TRVStyle 393
Select 558
TRVTableItemInfo 558
SelectAll 223
TRichView 223
SelectCols 558
TRVTableItemInfo 558
SelectControl 223
TRichView 223
SelectCurrentLine 337
TRichViewEdit 337
SelectCurrentWord 337
TRichViewEdit 337
Selecting parts of TRichView document 47
Selection in Table 116
SelectionExists 224
TRichView 224
SelectionHandleKind 398
TRVStyle 398
SelectionHandlesVisible 156
TRichView 156
SelectionMode 399
TRVStyle 399
SelectionStyle 399
TRVStyle 399
SelectRows 558
TRVTableItemInfo 558
SelectWordAt 224
TRichView 224
SelTextColor 393
TRVStyle 393
Semitransparent Objects in TRichView 57
SeqName 585
TRVSeqItemInfo 585
SetAddParagraphMode 225
TRichView 225
SetBackgroundImageEd 338
TRichViewEdit 338
SetBreakInfo 226
TRichView 226
SetBreakInfoEd 339
TRichViewEdit 339
SetBulletInfo 227
TRichView 227
SetBulletInfoEd 339
TRichViewEdit 339
SetButtonState 371
TRVSmartPopupProperties 371
SetCellBackgroundImage 559
TRVTableItemInfo 559
SetCellBackgroundImageFileName 559
TRVTableItemInfo 559
SetCellBackgroundStyle 559
TRVTableItemInfo 559
SetCellBestHeight 559
TRVTableItemInfo 559
SetCellBestWidth 560
TRVTableItemInfo 560
SetCellBorderColor 560
TRVTableItemInfo 560
SetCellBorderLightColor 560
TRVTableItemInfo 560
SetCellColor 560
TRVTableItemInfo 560
SetCellHint 561
TRVTableItemInfo 561
SetCellOpacity 561
TRVTableItemInfo 561
SetCellOptions 561
TRVTableItemInfo 561
SetCellRotation 561
TRVTableItemInfo 561
SetCellTag 562
TRVTableItemInfo 562
SetCellVAlign 562
TRVTableItemInfo 562
SetCellVisibleBorders 562
TRVTableItemInfo 562
SetCheckpointInfo 227
TRichView 227
SetCheckpointInfoEd 340
TRichViewEdit 340
SetControlInfo 228
TRichView 228
SetControlInfoEd 341

SetControlInfoEd 341
TRichViewEdit 341
SetCurrentBreakInfo 342
TRichViewEdit 342
SetCurrentBulletInfo 342
TRichViewEdit 342
SetCurrentCheckpointInfo 343
TRichViewEdit 343
SetCurrentControlInfo 344
TRichViewEdit 344
SetCurrentHotspotInfo 345
TRichViewEdit 345
SetCurrentItemExtraIntProperty 345
TRichViewEdit 345
SetCurrentItemExtraIntPropertyEx 345
TRichViewEdit 345
SetCurrentItemExtraStrProperty 346
TRichViewEdit 346
SetCurrentItemExtraStrPropertyEx 346
TRichViewEdit 346
SetCurrentItemText 347
TRichViewEdit 347
SetCurrentItemTextA 347
TRichViewEdit 347
SetCurrentItemTextW 347
TRichViewEdit 347
SetCurrentItemVAlign 348
TRichViewEdit 348
SetCurrentPictureInfo 348
TRichViewEdit 348
SetCurrentTag 349
TRichViewEdit 349
SetFloatPropertyEd 337
TRichViewEdit 337
SetFooter 276, 493
TRVPrint 493
TRVRTFReaderProperties 276
SetHeader 276, 493
TRVPrint 493
TRVRTFReaderProperties 276
SetHotspotInfo 229
TRichView 229
SetHotspotInfoEd 349
TRichViewEdit 349
SetIntPropertyEd 337
TRichViewEdit 337
SetItemExtraIntProperty 230
TRichView 230
SetItemExtraIntPropertyEd 350
TRichViewEdit 350
SetItemExtraIntPropertyEx 230
TRichView 230
SetItemExtraIntPropertyExEd 350
TRichViewEdit 350
SetItemExtraStrProperty 231
TRichView 231
SetItemExtraStrPropertyEd 351
TRichViewEdit 351
SetItemExtraStrPropertyEx 231
TRichView 231
SetItemExtraStrPropertyExEd 351
TRichViewEdit 351
SetItemTag 231
TRichView 231
SetItemTagEd 352
TRichViewEdit 352
SetItemText 232
TRichView 232
SetItemTextA 232
TRichView 232
SetItemTextEd 352
TRichViewEdit 352
SetItemTextEdA 352
TRichViewEdit 352
SetItemTextEdW 352
TRichViewEdit 352
SetItemTextW 232
TRichView 232
SetItemVAlign 233
TRichView 233
SetItemVAlignEd 353
TRichViewEdit 353
SetListMarkerInfo 233
TRichView 233
SetPictureInfo 234
TRichView 234
SetPictureInfoEd 354
TRichViewEdit 354
SetRowKeepTogether 562
TRVTableItemInfo 562
SetRowPageBreakBefore 562
TRVTableItemInfo 562
SetRowVAlign 562
TRVTableItemInfo 562
SetSelectionBounds 235
TRichView 235
SetStrPropertyEd 337
TRichViewEdit 337
SetTableVisibleBorders 563
TRVTableItemInfo 563

SetUndoGroupMode 355
TRichViewEdit 355
SetZoom 526
TCustomRVPrintPreview 526
ShadowColor 523
TCustomRVPrintPreview 523
ShadowWidth 523
TCustomRVPrintPreview 523
ShortCut 371
TRVSmartPopupProperties 371
SidenoteNumbering 389
TRVStyle 389
Sidenotes 105
SingleClick 156
TRichView 156
SingleSymbols 439
TCustomRVFontInfo 439
Size 441
TCustomRVFontInfo 441
SizeDouble 441
TCustomRVFontInfo 441
SkipHiddenText 273
TRVRTFReaderProperties 273
Smart Popups 368
SmartPopupProperties 295
TRichViewEdit 295
SmartPopupVisible 295
TRichViewEdit 295
Smooth image scaling in TRichView 57
SoftPageBreakColor 401
TRVStyle 401
SpaceAfter 454
TCustomRVParaInfo 454
SpaceBefore 455
TCustomRVParaInfo 455
SpacesInTab 402
TRVStyle 402
SpecialCharactersColor 402
TRVStyle 402
SplitSelectedCellsHorizontally 563
TRVTableItemInfo 563
SplitSelectedCellsVertically 563
TRVTableItemInfo 563
Standard 431
TCustomRVInfo 431
StartAnimation 237
TRichView 237
StartAt 517
TCustomRVPrint 517
StartFrom 479, 585
TRVListLevel 479
TRVSeqItemInfo 585
StartLiveSpelling 237
TRichView 237
StopAnimation 237
TRichView 237
StoreImagesFileNames 273
TRVRTFReaderProperties 273
Stream 500
TRVOfficeConverter 500
Style 156, 441, 468
TCustomRVFontInfo 441
TRichView 156
TRVBorder 468
style templates 457
StyleEx 441
TCustomRVFontInfo 441
StyleName 432
TCustomRVInfo 432
Styles and Style-Templates in TRichView 40
StyleTemplateId 432
TCustomRVFontOrParaInfo 432
StyleTemplateInsertMode 156
TRichView 156
StyleTemplates 402
TRVStyle 402
SubSuperScriptType 441
TCustomRVFontInfo 441

- T -

Table Cells 113
Table Operations 119
Table Resizing 124
Tables 98
Tables and Compatibility 112
Tables in RichView 111
Tables in TRichView 111
TabNavigation 157
TRichView 157
Tabs 455
TCustomRVParaInfo 455
Tabulators 89
Tag 574
TRVTableCellData 574
Tags of TRichView items 34
TBackgroundStyle 136
TCheckpointData 622
TConvertingEvent 503
TCPEventKind 139

TCustomRichView 128, 252
TCustomRichView.AssignSoftPageBreaks 174
TCustomRichView.GetItemAt 190
TCustomRichView.GetItemExtraStrProperty 193
TCustomRichView.GetItemExtraStrPropertyEx 193
TCustomRichView.OnReadHyperlink 248
TCustomRichViewEdit 282
TCustomRichViewEdit.ActualCurTextStyleNo 288
TCustomRVControl 508
TCustomRVData 608
TCustomRVData.Edit 610
TCustomRVData.GetRVData 611
TCustomRVData.GetSourceRVData 611
TCustomRVFieldItemInfo 604
TCustomRVFontInfo 432
TCustomRVFontInfo.Assign 443
TCustomRVFontInfo.AssignTo 443
TCustomRVFontInfo.BackColor 433
TCustomRVFontInfo.BiDiMode 433
TCustomRVFontInfo.CharScale 434
TCustomRVFontInfo.Charset 434
TCustomRVFontInfo.CharSpacing 435
TCustomRVFontInfo.Color 435
TCustomRVFontInfo.Create 444
TCustomRVFontInfo.FontName 436
TCustomRVFontInfo.HoverBackColor 436
TCustomRVFontInfo.HoverColor 436
TCustomRVFontInfo.HoverEffects 437
TCustomRVFontInfo.HoverUnderlineColor 437
TCustomRVFontInfo.IsEqual 444
TCustomRVFontInfo.JumpCursor 438
TCustomRVFontInfo.Options 438
TCustomRVFontInfo.ParaStyleTemplateId 439
TCustomRVFontInfo.Protection 439
TCustomRVFontInfo.SingleSymbols 439
TCustomRVFontInfo.Size 441
TCustomRVFontInfo.SizeDouble 441
TCustomRVFontInfo.Style 441
TCustomRVFontInfo.StyleEx 441
TCustomRVFontInfo.SubSuperScriptType 441
TCustomRVFontInfo.UnderlineColor 442
TCustomRVFontInfo.UnderlineType 442
TCustomRVFontInfo.VShift 443
TCustomRVFontOrParaInfo.StyleTemplateId 432
TCustomRVFormattedData 608
TCustomRVFormattedData.GetOriginEx 611
TCustomRVInfo 430
TCustomRVInfo.Standard 431
TCustomRVInfo.StyleName 432
TCustomRVIsURLFunction 662
TCustomRVItemInfo 526
TCustomRVNoteItemInfo 526, 586
TCustomRVNoteItemInfo.CreateEx 587
TCustomRVNoteItemInfo.Document 586
TCustomRVNoteItemInfo.NoteText 587
TCustomRVNoteItemInfo.ReplaceDocumentEd 587
TCustomRVPageNumberItemInfo 604
TCustomRVPageNumberItemInfo.NumberType 605
TCustomRVParaInfo 448
TCustomRVParaInfo.Alignment 449
TCustomRVParaInfo.Assign 455
TCustomRVParaInfo.Background 450
TCustomRVParaInfo.BiDiMode 450
TCustomRVParaInfo.Border 451
TCustomRVParaInfo.Create 455
TCustomRVParaInfo.FirstIndent 451
TCustomRVParaInfo.IsEqual 456
TCustomRVParaInfo.LastLineAlignment 449
TCustomRVParaInfo.LeftIndent 451
TCustomRVParaInfo.LineSpacing 452
TCustomRVParaInfo.LineSpacingType 452
TCustomRVParaInfo.Options 453
TCustomRVParaInfo.OutlineLevel 454
TCustomRVParaInfo.RightIndent 454
TCustomRVParaInfo.SpaceAfter 454
TCustomRVParaInfo.SpaceBefore 455
TCustomRVParaInfo.Tabs 455
TCustomRVPrint 514
TCustomRVPrint.Clear 517
TCustomRVPrint.ColorMode 514
TCustomRVPrint.EndAt 514
TCustomRVPrint.FirstPageNo 514
TCustomRVPrint.GetFirstItemOnPage 517
TCustomRVPrint.GetPageNo 518
TCustomRVPrint.IgnorePageBreaks 515
TCustomRVPrint.LastPageNo 515
TCustomRVPrint.MaxPrintedItemNo 515
TCustomRVPrint.MetafileCompatibility 515
TCustomRVPrint.MinPrintedItemNo 516
TCustomRVPrint.OnPrintComponent 519
TCustomRVPrint.PageNoFromNumber 516
TCustomRVPrint.PagesCount 516
TCustomRVPrint.PreviewCorrection 516
TCustomRVPrint.StartAt 517
TCustomRVPrint.TransparentBackground 517
TCustomRVPrint.UpdatePaletteInfo 518
TCustomRVPrintPreview 520
TCustomRVPrintPreview.ClickMode 521

TCustomRVPrintPreview.Color 521
TCustomRVPrintPreview.Create 525
TCustomRVPrintPreview.First 525
TCustomRVPrintPreview.Last 525
TCustomRVPrintPreview.MarginsPen 522
TCustomRVPrintPreview.Next 526
TCustomRVPrintPreview.OnZoomChanged 526
TCustomRVPrintPreview.PageBorderColor 522
TCustomRVPrintPreview.PageBorderWidth 522
TCustomRVPrintPreview.PageNo 522
TCustomRVPrintPreview.Prev 526
TCustomRVPrintPreview.PrintableAreaPen 523
TCustomRVPrintPreview.SetZoom 526
TCustomRVPrintPreview.ShadowColor 523
TCustomRVPrintPreview.ShadowWidth 523
TCustomRVPrintPreview.ZoomInCursor 523
TCustomRVPrintPreview.ZoomMode 524
TCustomRVPrintPreview.ZoomOutCursor 524
TCustomRVPrintPreview.ZoomPercent 525
TDBRichView 373
TDBRichView.AutoDisplay 374
TDBRichView.Create 375
TDBRichView.DataField 375
TDBRichView.DataSource 375
TDBRichView.Destroy 376
TDBRichView.Field 375
TDBRichView.LoadField 376
TDBRichView.OnLoadCustomFormat 376
TDBRichView.OnLoadDocument 376
TDBRichView.OnNewDocument 377
TDBRichViewEdit 377
  Inserting controls what can modify themselves in DBRichViewEdit 383
  Using viewer-style methods in DBRichViewEdit 383
TDBRichViewEdit.AutoDeleteUnusedStyles 378
TDBRichViewEdit.AutoDisplay 379
TDBRichViewEdit.Change 381
TDBRichViewEdit.Create 381
TDBRichViewEdit.DataField 379
TDBRichViewEdit.DataSource 379
TDBRichViewEdit.Destroy 381
TDBRichViewEdit.Field 379
TDBRichViewEdit.FieldFormat 380
TDBRichViewEdit.IgnoreEscape 380
TDBRichViewEdit.LoadField 381
TDBRichViewEdit.OnLoadCustomFormat 381
TDBRichViewEdit.OnLoadDocument 382
TDBRichViewEdit.OnNewDocument 382
TDBRichViewEdit.OnSaveCustomFormat 382
TDBRichViewEdit.ReadOnly 380
Text 581
  TRVLabelItemInfo 581
Text boxes 107
Text Items 88
TextBackgroundKind 402
  TRVStyle 402
TextColSeparator 543
  TRVTableItemInfo 543
TextEngine 403
  TRVStyle 403
TextRowSeparator 544
  TRVTableItemInfo 544
TextStyle 463
  TRVStyleTemplate 463
TextStyleMode 274
  TRVRTFReaderProperties 274
TextStyleNo 274, 581
  TRVLabelItemInfo 581
  TRVRTFReaderProperties 274
TextStyles 404
  TRVStyle 404
TFontInfo 444
TFontInfo.Assign 446
TFontInfo.Create 447
TFontInfo.Draw 447
TFontInfo.IsEqual 448
TFontInfo.Jump 445
TFontInfo.ModifiedProperties 445
TFontInfo.NextStyleNo 446
TFontInfo.Unicode 446
TFontInfos 418
TFontInfos.Add 419
TFontInfos.AddFont 419
TFontInfos.AddFontEx 419
TFontInfos.AssignTo 419
TFontInfos.Create 420
TFontInfos.FindStyleWithCharset 420
TFontInfos.FindStyleWithColor 420
TFontInfos.FindStyleWithFont 420
TFontInfos.FindStyleWithFontName 421
TFontInfos.FindStyleWithFontSize 421
TFontInfos.FindStyleWithFontStyle 421
TFontInfos.FindSuchStyle 422
TFontInfos.FindSuchStyleEx 422
TFontInfos.InvalidItem 418
TFontInfos.Items 418
TFontInfos.PixelsPerInch 419

Title 280
TRVDocParameters 280
TitlePage 280, 487
TRVDocParameters 280
TRVPrint 487
TJumpEvent 247
TopLevelEditor 295
TRichViewEdit 295
TopMargin 158, 280
TRichView 158
TRVDocParameters 280
TParaInfo 456
TParaInfo.Assign 458
TParaInfo.Create 458
TParaInfo.DefStyleNo 457
TParaInfo.IsEqual 458
TParaInfo.ModifiedProperties 457
TParaInfo.NextParaNo 457
TParaInfos 423
TParaInfos.Add 424
TParaInfos.AssignTo 424
TParaInfos.Create 424
TParaInfos.FindStyleWithAlignment 424
TParaInfos.FindSuchStyle 425
TParaInfos.FindSuchStyleEx 425
TParaInfos.InvalidItem 423
TParaInfos.Items 423
Tracking 511
TRVScroller 511
TransparentBackground 517
TCustomRVPrint 517
TRichView 128
adding items 43
Compatibility with older versions of RichView 705
Component Editor 134
live spelling check 69
Loading UTF-8 files 282
modifying items 51
New after v16 675
New in v1.2 704
New in v1.3 703
New in v1.4 702
New in v1.5 700
New in v1.6 698
New in v1.7 695
New in v1.8 693
New in v1.9 691
New in v10 687
New in v11 685
New in v12 684
New in v13 682
New in v14 679
New in v15 678
New in v16 676
obtaining items 50
selecting parts of document 47
Version history 675
TRichView Clipboard functions overview 48
TRichView constants and variables (List) 650
CFRV_HTML 651
CFRV_RTF 651
CFRV_RVF 651
CFRV_URL 651
HTMLFormatName 651
RichViewAllowCopyTableCells 652
RichViewAlternativePicPrint 652
RichViewAutoAssignNormalStyleTemplate 653
RichViewCompareStyleNames 653
RichViewDoNotCheckRVFStyleRefs 653
RichViewDoNotMergeNumbering 654
RichViewEditCaretHeightExtra 652
RichViewEditCaretWidth 652
RichViewEditDefaultProportionalResize 654
RichViewEditEnterAllowsEmptyMarkeredLines 654
RichViewEditMaxCaretHeight 652
RichViewJustifyBeforeLineBreak 654
RichViewMaxThumbnailCount 655
RichViewPixelsPerInch 655
RichViewResetStandardFlag 656
RichViewSaveDivInHTMLEx 652
RichViewSaveHyperlinksInDocXAsFields 656
RichViewSavePageBreaksInText 652
RichViewSavePInHTML 652
RichViewShowGhostSpaces 656
RichViewStringFormatMethod 656
RichViewTableAutoAddRow 657
RichViewTableDefaultRTFAutofit 657
RichViewUnicodeInput 657
RTFormatName 651
rveipcAlignment 658, 661
rveipcCursor 658, 661
rveipcMinWidth 658, 661
rveipcProtectTextStyleNo 658, 661
rveipcRemoveInternalLeading 658, 661
rveipcSeqNumberType 658, 661
rveipcSeqReset 658, 661
rveipcSeqStartFrom 658, 661
RVFormatName 651

TRichView constants and variables
RVIsCustomURL 662
RVParagraphMarks 662
rvsBreak 663
rvsBullet 663
rvsComponent 663
rvsDefStyle 663
rvsEndnote 663
rvsFootnote 663
rvsHeading 663
rvsHotPicture 663
rvsHotspot 663
rvsJump1 663
rvsJump2 663
rvsKeyword 663
rvsLabel 663
rvsListMarker 663
rvsNormal 663
rvsNoteReference 663
rvsPicture 663
rvsSequence 663
rvsSubheading 663
rvsTab 663
rvsTable 663
RVVisibleSpecialCharacters 665
URLFormatName 651
TRichView hypertext 35
TRichView item types 29, 85, 526
TRichView items' "checkpoints" overview 30
TRichView items' "tags" overview 34
TRichView methods for saving and loading 58
TRichView overview 26, 28
TRichView paragraphs 37
TRichView procedures and functions
DrawButton 616
DrawControl 616
DrawEdit 616
DrawMemo 616
DrawPanel 616
GetRVESearchOptions 616
GetRVSearchOptions 616
RV_GetPrinterDC 617
RV_TestFileUnicode 620
RV_TestStreamUnicode 620
RV_TestStringUnicode 620
RVGetFirstEndnote 618
RVGetFirstFootnote 618
RVGetFirstSidenote 618
RVGetNextEndnote 618
RVGetNextFootnote 618
RVGetNextSidenote 618
RVGetNoteTextStyleNo 619
RVIsEmail 619
RVIsURL 619
RVU_AnsiToUnicode 620
RVU_AnsiToUTF8 620
RVU_GetRawUnicode 620
RVU_RawUnicodeToWideString 620
RVU_UnicodeToAnsi 620
TRichView scrolling 46
TRichView.Add 167
TRichView.AddBreak 160
TRichView.AddBreakEx 160
TRichView.AddBreakExTag 160
TRichView.AddBreakTag 160
TRichView.AddBullet 160
TRichView.AddBulletEx 160
TRichView.AddBulletExTag 160
TRichView.AddCenterLine 161
TRichView.AddCheckpoint 166
TRichView.AddCheckpointTag 166
TRichView.AddControl 162
TRichView.AddControlEx 162
TRichView.AddControlExTag 162
TRichView.AddFmt 163
TRichView.AddFromNewLine 161
TRichView.AddHotPicture 163
TRichView.AddHotPictureTag 163
TRichView.AddHotspot 164
TRichView.AddHotspotEx 164
TRichView.AddHotspotExTag 164
TRichView.AddItem 166
TRichView.AddNamedCheckpoint 166
TRichView.AddNamedCheckpointEx 166
TRichView.AddNamedCheckpointExTag 166
TRichView.AddNL 167
TRichView.AddNLATag 167
TRichView.AddNLTag 167
TRichView.AddNLWTag 167
TRichView.AddPicture 169
TRichView.AddPictureEx 169
TRichView.AddPictureExTag 169
TRichView.AddTab 170
TRichView.AddTag 167
TRichView.AddText 171
TRichView.AddTextFromNewLine 171
TRichView.AddTextNL 171
TRichView.AddTextNLA 171
TRichView.AddTextNLW 171
TRichView.AllowSelection 135

TRichView.AnimationMode 135
TRichView.AppendFrom 172
TRichView.AppendRVFFromStream 173
TRichView.BackgroundBitmap 136
TRichView.BackgroundStyle 136
TRichView.BeginOleDrag 174
TRichView.BiDiMode 137
TRichView.BottomMargin 137
TRichView.Clear 175
TRichView.ClearLeft 138
TRichView.ClearLiveSpellingResults 175
TRichView.ClearRight 138
TRichView.ClearSoftPageBreaks 176
TRichView.ClientToDocument 176
TRichView.Color 139
TRichView.ConvertDocToDifferentUnits 176
TRichView.ConvertDocToPixels 176
TRichView.ConvertDocToTwips 176
TRichView.Copy 177
TRichView.CopyDef 177
TRichView.CopyImage 177
TRichView.CopyRTF 178
TRichView.CopyRVF 178
TRichView.CopyText 178
TRichView.CopyTextA 178
TRichView.CopyTextW 178
TRichView.CPEventKind 139
TRichView.Create 179
TRichView.Cursor 140
TRichView.DeleteItems 179
TRichView.DeleteMarkedStyles 180
TRichView.DeleteParas 180
TRichView.DeleteSection 180
TRichView.DeleteUnusedStyles 181
TRichView.Delimiters 140
TRichView.Deselect 181
TRichView.Destroy 181
TRichView.DocParameters 140
TRichView.DocProperties 141
TRichView.DocumentHeight 142
TRichView.DocumentToClient 176
TRichView.DoInPaletteMode 142
TRichView.FindCheckpointByName 182
TRichView.FindCheckpointByTag 182
TRichView.FindControlItemNo 182
TRichView.FirstItemVisible 144
TRichView.FirstJumpNo 144
TRichView.Format 182
TRichView.FormatTail 183
TRichView.GetBreakInfo 183
TRichView.GetBulletInfo 184
TRichView.GetCheckpointByNo 185
TRichView.GetCheckpointInfo 185
TRichView.GetCheckpointItemNo 186
TRichView.GetCheckpointNo 186
TRichView.GetCheckpointXY 186
TRichView.GetCheckpointY 186
TRichView.GetCheckpointYEx 187
TRichView.GetControlInfo 187
TRichView.GetFirstCheckpoint 188
TRichView.GetFocusedItem 188
TRichView.GetHotspotInfo 189
TRichView.GetItem 190
TRichView.GetItemCheckpoint 191
TRichView.GetItemClientCoords 191
TRichView.GetItemCoords 191
TRichView.GetItemCoordsEx 192
TRichView.GetItemExtraIntProperty 193
TRichView.GetItemExtraIntPropertyEx 193
TRichView.GetItemNo 194
TRichView.GetItemPara 194
TRichView.GetItemStyle 194
TRichView.GetItemTag 195
TRichView.GetItemText 195
TRichView.GetItemTextA 195
TRichView.GetItemTextW 195
TRichView.GetItemVAlign 196
TRichView.GetJumpPointItemNo 196
TRichView.GetJumpPointLocation 197
TRichView.GetJumpPointY 197
TRichView.GetLastCheckpoint 198
TRichView.GetLineNo 198
TRichView.GetListMarkerInfo 199
TRichView.GetNextCheckpoint 199
TRichView.GetOffsAfterItem 200
TRichView.GetOffsBeforeItem 201
TRichView.GetPictureInfo 201
TRichView.GetPrevCheckpoint 202
TRichView.GetSelectedImage 202
TRichView.GetSelectionBounds 203
TRichView.GetSelText 204
TRichView.GetSelTextA 204
TRichView.GetSelTextW 204
TRichView.GetTextInfo 204
TRichView.GetWordAt 205
TRichView.GetWordAtA 205
TRichView.GetWordAtW 205
TRichView.InsertRVFFromStream 206

TRichView.IsFromNewLine 207
TRichView.IsParaStart 208
TRichView.ItemCount 144
TRichView.LastItemVisible 144
TRichView.LeftMargin 145
TRichView.LineCount 145
TRichView.LiveSpellingValidateWord 208
TRichView.LoadFromStream 208
TRichView.LoadRTF 209
TRichView.LoadRTFFromStream 210
TRichView.LoadRVF 211
TRichView.LoadRVFFromStream 211
TRichView.LoadText 212
TRichView.LoadTextFromStream 213
TRichView.LoadTextFromStreamW 213
TRichView.LoadTextW 212
TRichView.MarkStylesInUse 213
TRichView.MaxLength 145
TRichView.MaxTextWidth 146
TRichView.MinTextWidth 146
TRichView.NoteText 146
TRichView.OnAddStyle 238
TRichView.OnCheckpointVisible 238
TRichView.OnControlAction 239
TRichView.OnCopy 240
TRichView.OnGetItemCursor 240
TRichView.OnHTMLSaveImage 241
TRichView.OnImportPicture 242
TRichView.OnItemAction 244
TRichView.OnItemHint 246
TRichView.OnJump 247
TRichView.OnPaint 247
TRichView.OnProgress 248
TRichView.OnRVDblClick 250
TRichView.OnRVFControlNeeded 251
TRichView.OnRVFImageListNeeded 252
TRichView.OnRVFPictureNeeded 252
TRichView.OnRVMouseDown 253
TRichView.OnRVMouseMove 254
TRichView.OnRVMouseUp 254
TRichView.OnRVRightClick 255
TRichView.OnSaveComponentToFile 256
TRichView.OnSaveDocXExtra 257
TRichView.OnSaveHTMLExtra 259
TRichView.OnSaveImage2 260
TRichView.OnSaveItemToFile 261
TRichView.OnSaveParaToHTML 263
TRichView.OnSaveRTFExtra 263
TRichView.OnSelect 265
TRichView.OnSpellingCheck 265
TRichView.OnStyleTemplatesChange 266
TRichView.OnURLNeeded 266
TRichView.OnWriteHyperlink 267
TRichView.Options 146
TRichView.PageBreaksBeforeItems 149
TRichView.Reformat 214
TRichView.RefreshListMarkers 214
TRichView.RefreshSequences 214
TRichView.RemoveCheckpoint 214
TRichView.ResetAnimation 215
TRichView.RightMargin 149
TRichView.RTFOptions 150
TRichView.RTFReadProperties 151
TRichView.RVData 151
TRichView.RVFOptions 152
TRichView.RVFParaStylesReadMode 154
TRichView.RVFTextStylesReadMode 154
TRichView.RVFWarnings 155
TRichView.SaveDocX 215
TRichView.SaveDocXToStream 216
TRichView.SaveHTML 216
TRichView.SaveHTMLEx 217
TRichView.SaveHTMLToStream 218
TRichView.SaveHTMLToStreamEx 218
TRichView.SavePicture 219
TRichView.SaveRTF 219
TRichView.SaveRTFToStream 220
TRichView.SaveRVF 220
TRichView.SaveRVFToStream 220
TRichView.SaveText 221
TRichView.SaveTextToStream 221
TRichView.SaveTextToStreamW 221
TRichView.SaveTextW 221
TRichView.SearchText 222
TRichView.SearchTextA 222
TRichView.SearchTextW 222
TRichView.SelectAll 223
TRichView.SelectControl 223
TRichView.SelectionExists 224
TRichView.SelectionHandlesVisible 156
TRichView.SelectWordAt 224
TRichView.SetAddParagraphMode 225
TRichView.SetBreakInfo 226
TRichView.SetBulletInfo 227
TRichView.SetCheckpointInfo 227
TRichView.SetControlInfo 228
TRichView.SetHotspotInfo 229
TRichView.SetItemExtraIntProperty 230

TRichView.SetItemExtraIntPropertyEx 230
TRichView.SetItemExtraStrProperty 231
TRichView.SetItemExtraStrPropertyEx 231
TRichView.SetItemTag 231
TRichView.SetItemText 232
TRichView.SetItemTextA 232
TRichView.SetItemTextW 232
TRichView.SetItemVAlign 233
TRichView.SetListMarkerInfo 233
TRichView.SetPictureInfo 234
TRichView.SetSelectionBounds 235
TRichView.SingleClick 156
TRichView.StartAnimation 237
TRichView.StartLiveSpelling 237
TRichView.StopAnimation 237
TRichView.Style 156
TRichView.StyleTemplateInsertMode 156
TRichView.TabNavigation 157
TRichView.TopMargin 158
TRichView.UpdatePaletteInfo 238
TRichView.UseStyleTemplates 158
TRichView.VAlign 159
TRichView.VSmallStep 159
TRichView.WordWrap 159
TRichViewEdit 282, 328
  Inserting items in position of caret 52
  InsertTextFromFile 328
  InsertTextFromFileW 328
  Moving caret to the beginning of paragraph 371
  OnDropFiles example 372
TRichViewEdit Undo 54
TRichViewEdit.AcceptDragDropFormats 287
TRichViewEdit.AcceptPasteFormats 287
TRichViewEdit.AdjustControlPlacement 296
TRichViewEdit.AdjustControlPlacement2 296
TRichViewEdit.ApplyListStyle 297
TRichViewEdit.ApplyParaStyle 297
TRichViewEdit.ApplyParaStyleConversion 298
TRichViewEdit.ApplyParaStyleTemplate 298
TRichViewEdit.ApplyStyleConversion 299
TRichViewEdit.ApplyStyleTemplate 299
TRichViewEdit.ApplyTextStyle 300
TRichViewEdit.ApplyTextStyleTemplate 301
TRichViewEdit.BeginItemModify 301
TRichViewEdit.BeginUndoCustomGroup 302
TRichViewEdit.BeginUndoGroup 302
TRichViewEdit.BeginUpdate 302
TRichViewEdit.CanChange 302
TRichViewEdit.CanPaste 303
TRichViewEdit.CanPasteRTF 303
TRichViewEdit.CanPasteRVF 303
TRichViewEdit.Change 303
TRichViewEdit.ChangeListLevels 304
TRichViewEdit.ChangeStyleTemplates 304
TRichViewEdit.Clear 304
TRichViewEdit.ClearTextFlow 305
TRichViewEdit.ClearUndo 305
TRichViewEdit.ConvertToHotPicture 305
TRichViewEdit.ConvertToPicture 305
TRichViewEdit.Create 306
TRichViewEdit.CurItemNo 288
TRichViewEdit.CurItemStyle 289
TRichViewEdit.CurParaStyleNo 289
TRichViewEdit.CurTextStyleNo 290
TRichViewEdit.CustomCaretInterval 290
TRichViewEdit.CutDef 306
TRichViewEdit.DefaultPictureVAlign 291
TRichViewEdit.DeleteSelection 306
TRichViewEdit.Destroy 306
TRichViewEdit.EditorOptions 291
TRichViewEdit.EndItemModify 307
TRichViewEdit.EndUpdate 307
TRichViewEdit.ForceFieldHighlight 293
TRichViewEdit.GetCheckpointAtCaret 307
TRichViewEdit.GetCurrentBreakInfo 308
TRichViewEdit.GetCurrentBulletInfo 308
TRichViewEdit.GetCurrentCheckpoint 309
TRichViewEdit.GetCurrentControlInfo 309
TRichViewEdit.GetCurrentHotspotInfo 310
TRichViewEdit.GetCurrentItem 311
TRichViewEdit.GetCurrentItemEx 311
TRichViewEdit.GetCurrentItemExtraIntProperty 312
TRichViewEdit.GetCurrentItemExtraIntPropertyEx 312
TRichViewEdit.GetCurrentItemExtraStrProperty 313
TRichViewEdit.GetCurrentItemExtraStrPropertyEx 313
TRichViewEdit.GetCurrentItemText 313
TRichViewEdit.GetCurrentItemTextA 313
TRichViewEdit.GetCurrentItemTextW 313
TRichViewEdit.GetCurrentItemVAlign 314
TRichViewEdit.GetCurrentLineCol 314
TRichViewEdit.GetCurrentMisspelling 315
TRichViewEdit.GetCurrentPictureInfo 315
TRichViewEdit.GetCurrentTag 316
TRichViewEdit.GetCurrentTextInfo 316
TRichViewEdit.InsertBreak 317
TRichViewEdit.InsertBullet 317

TRichViewEdit.InsertCheckpoint 318
TRichViewEdit.InsertControl 318
TRichViewEdit.InsertHotPicture 319
TRichViewEdit.InsertHotspot 319
TRichViewEdit.InsertItem 320
TRichViewEdit.InsertOEMTextFromFile 321
TRichViewEdit.InsertPageBreak 321
TRichViewEdit.InsertPicture 322
TRichViewEdit.InsertRTFFromFileEd 323
TRichViewEdit.InsertRTFFromStreamEd 323
TRichViewEdit.InsertRVFFromFileEd 324
TRichViewEdit.InsertRVFFromStreamEd 325
TRichViewEdit.InsertStringATag 326
TRichViewEdit.InsertStringTag 326
TRichViewEdit.InsertStringWTag 326
TRichViewEdit.InsertTab 326
TRichViewEdit.InsertText 327
TRichViewEdit.InsertTextA 327
TRichViewEdit.InsertTextW 327
TRichViewEdit.LiveSpellingMode 293
TRichViewEdit.Modified 293
TRichViewEdit.OffsetInCurItem 294
TRichViewEdit.OnCaretGetOut 356
TRichViewEdit.OnCaretMove 356
TRichViewEdit.OnChange 357
TRichViewEdit.OnChanging 357
TRichViewEdit.OnCheckStickingItems 357
TRichViewEdit.OnCurParaStyleChanged 358
TRichViewEdit.OnCurTextStyleChanged 358
TRichViewEdit.OnDrawCustomCaret 358
TRichViewEdit.OnDropFiles 359
TRichViewEdit.OnItemResize 359
TRichViewEdit.OnItemTextEdit 360
TRichViewEdit.OnMeasureCustomCaret 360
TRichViewEdit.OnOleDragEnter 361
TRichViewEdit.OnOleDragLeave 362
TRichViewEdit.OnOleDragOver 362
TRichViewEdit.OnOleDrop 363
TRichViewEdit.OnParaStyleConversion 364
TRichViewEdit.OnPaste 365
TRichViewEdit.OnSmartPopupClick 366
TRichViewEdit.OnStyleConversion 366
TRichViewEdit.Paste 328
TRichViewEdit.PasteBitmap 329
TRichViewEdit.PasteGraphicFile 329
TRichViewEdit.PasteMetafile 330
TRichViewEdit.PasteRTF 330
TRichViewEdit.PasteRVF 331
TRichViewEdit.PasteText 331
TRichViewEdit.PasteTextA 331
TRichViewEdit.PasteTextW 331
TRichViewEdit.PasteURL 332
TRichViewEdit.ReadOnly 294
TRichViewEdit.Redo 332
TRichViewEdit.RedoAction 333
TRichViewEdit.RedoName 333
TRichViewEdit.RemoveCheckpointAtCaret 333
TRichViewEdit.RemoveCheckpointEd 333
TRichViewEdit.RemoveCurrentCheckpoint 334
TRichViewEdit.RemoveCurrentPageBreak 334
TRichViewEdit.RemoveLists 334
TRichViewEdit.ResizeControl 335
TRichViewEdit.ResizeCurrentControl 335
TRichViewEdit.SearchText 336
TRichViewEdit.SearchTextA 336
TRichViewEdit.SearchTextW 336
TRichViewEdit.SelectCurrentLine 337
TRichViewEdit.SelectCurrentWord 337
TRichViewEdit.SetBackgroundImageEd 338
TRichViewEdit.SetBreakInfoEd 339
TRichViewEdit.SetBulletInfoEd 339
TRichViewEdit.SetCheckpointInfoEd 340
TRichViewEdit.SetControlInfoEd 341
TRichViewEdit.SetCurrentBreakInfo 342
TRichViewEdit.SetCurrentBulletInfo 342
TRichViewEdit.SetCurrentCheckpointInfo 343
TRichViewEdit.SetCurrentControlInfo 344
TRichViewEdit.SetCurrentHotspotInfo 345
TRichViewEdit.SetCurrentItemExtraIntProperty 345
TRichViewEdit.SetCurrentItemExtraIntPropertyEx 345
TRichViewEdit.SetCurrentItemExtraStrProperty 346
TRichViewEdit.SetCurrentItemExtraStrPropertyEx 346
TRichViewEdit.SetCurrentItemText 347
TRichViewEdit.SetCurrentItemTextA 347
TRichViewEdit.SetCurrentItemTextW 347
TRichViewEdit.SetCurrentItemVAlign 348
TRichViewEdit.SetCurrentPictureInfo 348
TRichViewEdit.SetCurrentTag 349
TRichViewEdit.SetFloatPropertyEd 337
TRichViewEdit.SetHotspotInfoEd 349
TRichViewEdit.SetIntPropertyEd 337
TRichViewEdit.SetItemExtraIntPropertyEd 350
TRichViewEdit.SetItemExtraIntPropertyExEd 350
TRichViewEdit.SetItemExtraStrPropertyEd 351
TRichViewEdit.SetItemExtraStrPropertyExEd 351
TRichViewEdit.SetItemTagEd 352

TRichViewEdit.SetItemTextEd 352
TRichViewEdit.SetItemTextEdA 352
TRichViewEdit.SetItemTextEdW 352
TRichViewEdit.SetItemVAlignEd 353
TRichViewEdit.SetPictureInfoEd 354
TRichViewEdit.SetStrPropertyEd 337
TRichViewEdit.SetUndoGroupMode 355
TRichViewEdit.SmartPopupProperties 295
TRichViewEdit.SmartPopupVisible 295
TRichViewEdit.TopLevelEditor 295
TRichViewEdit.Undo 355
TRichViewEdit.UndoAction 355
TRichViewEdit.UndoLimit 295
TRichViewEdit.UndoName 355
TRichViewRVData 608
TRichViewUnicodeInput 657
TRV_AfterImportGraphicsProc 671
TRV_CreateGraphicsFunction 671
TRVAddStyleEvent 238
TRVAfterApplyStyleEvent 410
TRVAlignment 449
TRVAnimationMode 135
TRVAnsiString 623
TRVApplyStyleColorEvent 411
TRVApplyStyleEvent 410
TRVBackgroundRect 466
TRVBackgroundRect.BorderOffsets 466
TRVBackgroundRect.Color 467
TRVBackgroundRect.Opacity 467
TRVBiDiMode 623
TRVBooleanRect 611
TRVBorder 467
TRVBorder.BorderOffsets 467
TRVBorder.Color 468
TRVBorder.InternalWidth 468
TRVBorder.Style 468
TRVBorder.VisibleBorders 468
TRVBorder.Width 468
TRVBorderStyle 623
TRVBoxBackgroundRect 594
TRVBoxBorder 594
TRVBoxHeightType 594
TRVBoxPosition 596
TRVBoxPosition.HorizontalAlignment 597
TRVBoxPosition.HorizontalAnchor 596
TRVBoxPosition.HorizontalOffset 597
TRVBoxPosition.HorizontalPositionKind 597
TRVBoxPosition.PositionInText 599
TRVBoxPosition.RelativeToCell 599
TRVBoxPosition.VerticalAlignment 600
TRVBoxPosition.VerticalAnchor 599
TRVBoxPosition.VerticalOffset 600
TRVBoxPosition.VerticalPositionKind 600
TRVBoxProperties 593
TRVBoxProperties.Background 594
TRVBoxProperties.Border 594
TRVBoxProperties.Height 594
TRVBoxProperties.HeightType 594
TRVBoxProperties.VAlign 595
TRVBoxProperties.Width 595
TRVBoxProperties.WidthType 595
TRVBoxWidthType 595
TRVBreakItemInfo 526
TRVBreakStyle 624
TRVBulletItemInfo 526
TRVCellEditingEvent 564
TRVCellVAlign 625
TRVCheckPointVisibleEvent 238
TRVClickMode 521
TRVCodePage 620, 625
TRVColorMode 514, 625
TRVControlAction 239
TRVControlActionEvent 239
TRVControlData 608
TRVControlItemInfo 526
TRVCustomFormatEvent 376, 381, 382
TRVDBFieldFormat 380
TRVDblClickEvent 250
TRVDeleteUnusedStylesData 612
TRVDisplayOption 626
TRVDisplayOptions 626
TRVDocParameters 140, 277
TRVDocParameters.Author 277
TRVDocParameters.BottomMargin 277
TRVDocParameters.Comments 278
TRVDocParameters.ConvertToUnit 281
TRVDocParameters.FacingPages 278
TRVDocParameters.FooterY 278
TRVDocParameters.HeaderY 278
TRVDocParameters.LeftMargin 278
TRVDocParameters.MirrorMargins 279
TRVDocParameters.Orientation 279
TRVDocParameters.PageHeight 279
TRVDocParameters.PageWidth 279
TRVDocParameters.Reset 281
TRVDocParameters.ResetLayout 281
TRVDocParameters.RightMargin 279
TRVDocParameters.Title 280

TRVDocParameters.TitlePage 280
TRVDocParameters.TopMargin 280
TRVDocParameters.Units 280
TRVDocParameters.UnitsPerInch 281
TRVDocParameters.ZoomMode 281
TRVDocParameters.ZoomPercent 281
TRVDocRotation 626
TRVDocXSaveArea 257
TRVDragDropFormat 287
TRVDragDropFormats 287
TRVDrawCheckpointEvent 411, 507
TRVDrawCustomCaretEvent 358
TRVDrawHyperlinkEvent 508
TRVDrawPageBreakEvent 413
TRVDrawParaRectEvent 414
TRVDrawStyleTextEvent 415
TRVDrawTextBackEvent 416
TRVDropFileAction 359
TRVDropFilesEvent 359
TRVEditRVData 608
TRVEndnoteItemInfo 526, 587
TRVEndnoteItemInfo.Create 588
TRVEndnoteItemInfo.CreateEx 589
TRVESearchOption 626
TRVESearchOptions 626
TRVExtraItemProperty 627
TRVExtraItemStrProperty 630
TRVFControlNeededEvent 251
TRVFImageListNeededEvent 252
TRVFixMarginsMode 484
TRVFloatHorizontalAlignment 597
TRVFloatPosition 632
TRVFloatPositionInText 599
TRVFloatPositionKind 632
TRVFloatProperty 632
TRVFloatSize 633
TRVFloatVerticalAlignment 600
TRVFontInfoClass 407
TRVFontInfoProperties 633
TRVFontInfoProperty 633
TRVFontStyle 441
TRVFontStyles 441
TRVFootnoteItemInfo 526, 589, 591
TRVFootnoteItemInfo.Create 590
TRVFootnoteItemInfo.CreateEx 591
TRVFootOrEndnoteItemInfo 586
TRVFOption 152
TRVFOptions 152
TRVFPictureNeededEvent 252
TRVFReaderStyleMode 633
TRVFullLineItemInfo 526
TRVFWarning 155
TRVFWarnings 155
TRVGetItemCursorEvent 240
TRVGetOutDirection 356
TRVGetPageNumberEvent 508
TRVGraphicHandler 613
TRVGraphicItemInfo 526
TRVHFType 634
TRVHorizontalAnchor 596
TRVHotGraphicItemInfo 526
TRVHotspotItemInfo 526
TRVHoverEffect 437
TRVHoverEffects 437
TRVHTMLLength 635
TRVHTMLSaveArea 259
TRVHTMLSaveImageEvent 241
TRVImportPictureEvent 242
TRVIntegerList 614
TRVIntProperty 635
TRVItemAction 244
TRVItemActionEvent 244
TRVItemBackgroundStyle 636
TRVItemFormattedData 608
TRVItemHintEvent 246
TRVItemResizeEvent 359
TRVItemTextEditEvent 360
TRVLabelItemInfo 526, 579
TRVLabelItemInfo.Alignment 580
TRVLabelItemInfo.Create 582
TRVLabelItemInfo.CreateEx 582
TRVLabelItemInfo.Cursor 580
TRVLabelItemInfo.MinWidth 580
TRVLabelItemInfo.ProtectTextStyleNo 581
TRVLabelItemInfo.RemoveInternalLeading 581
TRVLabelItemInfo.Text 581
TRVLabelItemInfo.TextStyleNo 581
TRVLastLineAlignment 449
TRVLength 636
TRVLineSpacingType 452
TRVLineSpacingValue 452
TRVLineWrapMode 394
TRVListInfo 459
TRVListInfo.AllNumbered 459
TRVListInfo.HasNumbering 460
TRVListInfo.Levels 459
TRVListInfo.OneLevelPreview 459
TRVListInfoClass 406

TRVListInfos 425
TRVListInfos.Add 426
TRVListInfos.AssignTo 426
TRVListInfos.FindStyleWithLevels 426
TRVListInfos.FindSuchStyle 426
TRVListInfos.Items 426
TRVListLevel 472
TRVListLevel.FirstIndent 473
TRVListLevel.Font 473
TRVListLevel.FormatString 474
TRVListLevel.FormatStringW 475
TRVListLevel.HasNumbering 479
TRVListLevel.ImageIndex 475
TRVListLevel.ImageList 475
TRVListLevel.LeftIndent 476
TRVListLevel.ListType 476
TRVListLevel.MarkerAlignment 477
TRVListLevel.MarkerIndent 478
TRVListLevel.Options 478
TRVListLevel.Picture 479
TRVListLevel.StartFrom 479
TRVListLevelCollection 471
TRVListLevelCollection.Add 472
TRVListLevelCollection.Items 472
TRVListLevelOption 478
TRVListLevelOptions 478
TRVListType 476
TRVLiveSpellingMode 293
TRVLoadFormat 248
TRVLongOperation 248
TRVMarginsPen 522
TRVMarkerAlignment 477
TRVMarkerItemInfo 526
TRVMeasureCustomCaretEvent 360
TRVMouseEvent 253, 254
TRVMouseMoveEvent 254
TRVNonTextItemInfo 526
TRVNoteData 608
TRVNoteReferenceItemInfo 526, 602
TRVNoteReferenceItemInfo.Create 604
TRVNoteReferenceItemInfo.CreateEx 604
TRVOfficeCnvList 503
TRVOfficeCnvList.Count 503
TRVOfficeCnvList.Items 503
TRVOfficeConverter 496
TRVOfficeConverter.Create 500
TRVOfficeConverter.Destroy 500
TRVOfficeConverter.ErrorCode 497
TRVOfficeConverter.ExcludeDocXExportConverter 498
TRVOfficeConverter.ExcludeDocXImportConverter 498
TRVOfficeConverter.ExcludeHTMLExportConverter 498
TRVOfficeConverter.ExcludeHTMLImportConverter 498
TRVOfficeConverter.ExportConverters 499
TRVOfficeConverter.ExportRTF 500
TRVOfficeConverter.ExportRV 500
TRVOfficeConverter.ExtensionsInFilter 499
TRVOfficeConverter.GetExportFilter 501
TRVOfficeConverter.GetImportFilter 501
TRVOfficeConverter.ImportConverters 499
TRVOfficeConverter.ImportRTF 501
TRVOfficeConverter.ImportRV 502
TRVOfficeConverter.IsValidImporter 502
TRVOfficeConverter.OnConverting 503
TRVOfficeConverter.PreviewMode 499
TRVOfficeConverter.Stream 500
TRVOfficeConverterInfo 503
TRVOfficeConverterInfo.Filter 503
TRVOfficeConverterInfo.Name 503
TRVOfficeConverterInfo.Path 503
TRVOleDragEnterEvent 361
TRVOleDragOverEvent 362
TRVOleDropEffect 636
TRVOleDropEffects 636
TRVOleDropEvent 363
TRVOnCaretGetOutEvent 356
TRVOpacity 637
TRVOption 146
TRVOptions 146
TRVPageBreakType 413
TRVPageCountItemInfo 607
TRVPageCountItemInfo.Create 607
TRVPageCountItemInfo.CreateEx 607
TRVPageNumberItemInfo 605
TRVPageNumberItemInfo.Create 606
TRVPageNumberItemInfo.CreateEx 606
TRVPageNumberType 637
TRVPagePrepaintEvent 495
TRVPaintEvent 247
TRVPaletteAction 142
TRVPAPen 523
TRVParagraphMarks 662
TRVParaInfoClass 407
TRVParaInfoProperties 637

TRVParaInfoProperty 637
TRVParaOption 453
TRVParaOptions 453
TRVPasteEvent 365
TRVPrint 479
TRVPrint.AssignDocParameters 488
TRVPrint.AssignSource 488
TRVPrint.ClipMargins 484
TRVPrint.ContinuousPrint 488
TRVPrint.ConvertToUnits 489
TRVPrint.Create 489
TRVPrint.DrawPreview 489
TRVPrint.FacingPages 484
TRVPrint.FixMarginsMode 484
TRVPrint.FooterY 485
TRVPrint.FormatPages 490
TRVPrint.GetFooterRect 490
TRVPrint.GetHeaderRect 490
TRVPrint.HeaderY 485
TRVPrint.MakePreview 491
TRVPrint.MakeScaledPreview 491
TRVPrint.Margins 486
TRVPrint.MirrorMargins 487
TRVPrint.OnFormatting 494
TRVPrint.OnPagePostpaint 495
TRVPrint.OnPagePrepaint 495
TRVPrint.OnSendingToPrinter 496
TRVPrint.Preview100PercentHeight 487
TRVPrint.Preview100PercentWidth 487
TRVPrint.Print 492
TRVPrint.PrintPages 492
TRVPrint.SetFooter 493
TRVPrint.SetHeader 493
TRVPrint.TitlePage 487
TRVPrint.Units 487
TRVPrintComponentEvent 519
TRVPrintingEvent 494, 496
TRVPrintingStep 638
TRVPrintPreview 384
TRVPrintPreview.RVPrint 384
TRVProgressEvent 248
TRVProgressStage 248
TRVProtectOption 439
TRVProtectOptions 439
TRVRawByteString 638
TRVReaderStyleMode 639
TRVReaderUnicode 274
TRVReadHyperlink 248
TRVRect 615
TRVRectItemInfo 526
TRVReportHelper 503
TRVReportHelper.AllowTransparentDrawing 504
TRVReportHelper.DrawPage 505
TRVReportHelper.DrawPageAt 506
TRVReportHelper.Finished 506
TRVReportHelper.FormatNextPage 506
TRVReportHelper.GetLastPageHeight 506
TRVReportHelper.Init 507
TRVReportHelper.NoMetafiles 505
TRVReportHelper.OnDrawCheckpoint 507
TRVReportHelper.OnDrawHyperlink 508
TRVReportHelper.OnGetPageNumber 508
TRVReportHelper.RichView 505
TRVReportHelper.UpdatePageNumbers 507
TRVRightClickEvent 255
TRVRTFErrorCode 270
TRVRTFHighlight 270
TRVRTFOption 150
TRVRTFOptions 150
TRVRTFReaderProperties 268
TRVRTFReaderProperties.AutoHideTableGridLines 269
TRVRTFReaderProperties.BasePathLinks 269
TRVRTFReaderProperties.CharsetForUnicode 269
TRVRTFReaderProperties.ConvertHighlight 270
TRVRTFReaderProperties.ErrorCode 270
TRVRTFReaderProperties.ExtractMetafileBitmaps 271
TRVRTFReaderProperties.IgnoreBookmarks 271
TRVRTFReaderProperties.IgnoreFields 271
TRVRTFReaderProperties.IgnoreNotes 271
TRVRTFReaderProperties.IgnorePictures 271
TRVRTFReaderProperties.IgnoreSequences 272
TRVRTFReaderProperties.IgnoreTables 272
TRVRTFReaderProperties.LineBreaksAsParagraphs 272
TRVRTFReaderProperties.ParaStyleMode 272
TRVRTFReaderProperties.ParaStyleNo 273
TRVRTFReaderProperties.ReadDocParameters 273
TRVRTFReaderProperties.SetFooter 276
TRVRTFReaderProperties.SetHeader 276
TRVRTFReaderProperties.SkipHiddenText 273
TRVRTFReaderProperties.StoreImagesFileNames 273
TRVRTFReaderProperties.TextStyleMode 274
TRVRTFReaderProperties.TextStyleNo 274
TRVRTFReaderProperties.UnicodeMode 274

TRVRTFReaderProperties.UseCharsetForUnicode 275
TRVRTFReaderProperties.UseHypertextStyles 275
TRVRTFReaderProperties.UseSingleCellPadding 275
TRVRTFSaveArea 263
TRVSaveComponentToFileEvent 256
TRVSaveCSSOption 408
TRVSaveCSSOptions 408
TRVSaveDocXExtraEvent 257
TRVSaveFormat 639
TRVSaveHTMLExtraEvent 259
TRVSaveImageEvent2 260
TRVSaveOption 639
TRVSaveOptions 639
TRVSaveParaToHTMLEvent 263
TRVSaveRTFExtraEvent 263
TRVScroller 508
TRVScroller.BorderStyle 509
TRVScroller.Create 513
TRVScroller.HScrollMax 509
TRVScroller.HScrollPos 510
TRVScroller.HScrollVisible 510
TRVScroller.InplaceEditor 510
TRVScroller.NoVScroll 511
TRVScroller.OnHScrolled 513
TRVScroller.OnVScrolled 513
TRVScroller.ScrollTo 513
TRVScroller.Tracking 511
TRVScroller.UseXPThemes 511
TRVScroller.VScrollMax 512
TRVScroller.VScrollPos 512
TRVScroller.VScrollVisible 512
TRVScroller.WheelStep 512
TRVSearchOption 642
TRVSearchOptions 642
TRVSelection 621
TRVSelectionHandleKind 398
TRVSelectionMode 399
TRVSelectionStyle 399
TRVSeqItemInfo 526, 582
TRVSeqItemInfo.Create 585
TRVSeqItemInfo.CreateEx 585
TRVSeqItemInfo.FormatString 584
TRVSeqItemInfo.NumberType 584
TRVSeqItemInfo.Reset 584
TRVSeqItemInfo.SeqName 585
TRVSeqItemInfo.StartFrom 585
TRVSeqType 643
TRVSidenoteItemInfo.BoxPosition 592
TRVSidenoteItemInfo.BoxProperties 592
TRVSidenoteItemInfo.Create 592
TRVSidenoteItemInfo.CreateEx 593
TRVSmartPopupClickEvent 366
TRVSmartPopupPosition 371
TRVSmartPopupProperties 368
TRVSmartPopupProperties.ButtonType 369
TRVSmartPopupProperties.Color 369
TRVSmartPopupProperties.Hint 370
TRVSmartPopupProperties.HoverColor 370
TRVSmartPopupProperties.HoverLineColor 370
TRVSmartPopupProperties.ImageIndex 370
TRVSmartPopupProperties.ImageList 370
TRVSmartPopupProperties.LineColor 370
TRVSmartPopupProperties.Menu 371
TRVSmartPopupProperties.Position 371
TRVSmartPopupProperties.SetButtonState 371
TRVSmartPopupProperties.ShortCut 371
TRVSpecialCharacter 665
TRVSpecialCharacters 665
TRVSpellingCheckEvent 265
TRVSTFontInfo 463
TRVStickingItemsEvent 357
TRVSTParaInfo 463
TRVStringFormatMethod 656
TRVStrProperty 643
TRVStyle 385
TRVStyle.AddTextStyle 405
TRVStyle.CheckpointColor 387
TRVStyle.CheckpointEvColor 387
TRVStyle.Color 387
TRVStyle.ConvertToDifferentUnits 405
TRVStyle.ConvertToPixels 405
TRVStyle.ConvertToTwips 405
TRVStyle.Create 405
TRVStyle.CurrentItemColor 388
TRVStyle.DefCodePage 388
TRVStyle.DefTabWidth 388
TRVStyle.DefUnicodeStyle 389
TRVStyle.DeleteTextStyle 405
TRVStyle.Destroy 405
TRVStyle.DifferentUnitsToUnits 406
TRVStyle.DisabledFontColor 389
TRVStyle.EndnoteNumbering 389
TRVStyle.FieldHighlightColor 390
TRVStyle.FieldHighlightType 390
TRVStyle.FindParaStyle 407
TRVStyle.FindTextStyle 407

TRVStyle.FloatingLineColor 390
TRVStyle.FontQuality 391
TRVStyle.FootnoteNumbering 389
TRVStyle.FootnotePageReset 392
TRVStyle.FullRedraw 392
TRVStyle.GetAsDifferentUnits 406
TRVStyle.GetAsPixels 406
TRVStyle.GetAsRVUnits 406
TRVStyle.GetAsTwips 406
TRVStyle.GetListStyleClass 406
TRVStyle.GetParaStyleClass 407
TRVStyle.GetTextStyleClass 407
TRVStyle.GridColor 392
TRVStyle.GridReadOnlyColor 392
TRVStyle.GridReadOnlyStyle 392
TRVStyle.GridStyle 392
TRVStyle.HoverColor 393
TRVStyle.InactiveSelColor 393
TRVStyle.InactiveSelTextColor 393
TRVStyle.InvalidPicture 393
TRVStyle.JumpCursor 394
TRVStyle.LineSelectCursor 394
TRVStyle.LineWrapMode 394
TRVStyle.ListStyles 396
TRVStyle.LiveSpellingColor 397
TRVStyle.LoadINI 407
TRVStyle.LoadReg 408
TRVStyle.MainRVStyle 397
TRVStyle.OnAfterApplyStyle 410
TRVStyle.OnApplyStyle 410
TRVStyle.OnApplyStyleColor 411
TRVStyle.OnDrawCheckpoint 411
TRVStyle.OnDrawPageBreak 413
TRVStyle.OnDrawParaBack 414
TRVStyle.OnDrawStyleText 415
TRVStyle.OnDrawTextBack 416
TRVStyle.OnStyleHoverSensitive 417
TRVStyle.PageBreakColor 398
TRVStyle.ParaStyles 398
TRVStyle.PixelsToUnits 406
TRVStyle.RVUnitsToUnits 406
TRVStyle.SaveCSS 408
TRVStyle.SaveCSSToStream 409
TRVStyle.SaveINI 409
TRVStyle.SaveReg 409
TRVStyle.SelColor 393
TRVStyle.SelectionHandleKind 398
TRVStyle.SelectionMode 399
TRVStyle.SelectionStyle 399
TRVStyle.SelTextColor 393
TRVStyle.SidenoteNumbering 389
TRVStyle.SoftPageBreakColor 401
TRVStyle.SpacesInTab 402
TRVStyle.SpecialCharactersColor 402
TRVStyle.StyleTemplates 402
TRVStyle.TextBackgroundKind 402
TRVStyle.TextEngine 403
TRVStyle.TextStyles 404
TRVStyle.TwipsToUnits 406
TRVStyle.Units 404
TRVStyle.UseSound 404
TRVStyleConversionEvent 364, 366
TRVStyleHoverSensitiveEvent 417
TRVStyleLength 644
TRVStyleTemplate 460
TRVStyleTemplate.ApplyToParaStyle 464
TRVStyleTemplate.ApplyToTextStyle 465
TRVStyleTemplate.Id 461
TRVStyleTemplate.Kind 461
TRVStyleTemplate.Name 461
TRVStyleTemplate.Next 462
TRVStyleTemplate.NextId 462
TRVStyleTemplate.ParaStyle 463
TRVStyleTemplate.Parent 463
TRVStyleTemplate.ParentId 463
TRVStyleTemplate.QuickAccess 463
TRVStyleTemplate.TextStyle 463
TRVStyleTemplate.UpdateModifiedParaStyleProperties 466
TRVStyleTemplate.UpdateModifiedTextStyleProperties 466
TRVStyleTemplate.ValidParaProperties 463
TRVStyleTemplate.ValidTextProperties 464
TRVStyleTemplateCollection 427
TRVStyleTemplateCollection.AssignStyleTemplates 428
TRVStyleTemplateCollection.AssignToStrings 428
TRVStyleTemplateCollection.ClearParaFormat 429
TRVStyleTemplateCollection.ClearTextFormat 429
TRVStyleTemplateCollection.DefStyleName 427
TRVStyleTemplateCollection.FindById 429
TRVStyleTemplateCollection.FindByName 429
TRVStyleTemplateCollection.FindItemById 429
TRVStyleTemplateCollection.FindItemByName 429
TRVStyleTemplateCollection.ForbiddenIds 428
TRVStyleTemplateCollection.Items 428
TRVStyleTemplateCollection.LoadFromRVST 429

TRVStyleTemplateCollection.NormalStyleTemplate 428
TRVStyleTemplateCollection.ResetNameCounter 430
TRVStyleTemplateCollection.SaveToRVST 429
TRVStyleTemplateId 644
TRVStyleTemplateInsertMode 156
TRVStyleTemplateKind 461
TRVStyleTemplateName 644
TRVStyleUnits 644
TRVSubData 608
TRVSubSuperScriptType 441
TRVTabAlign 470
TRVTabInfo 470
TRVTabInfo.Align 470
TRVTabInfo.Leader 470
TRVTabInfo.Position 471
TRVTabInfos 469
TRVTabInfos.AddFrom 469
TRVTabInfos.DeleteList 469
TRVTabInfos.Find 469
TRVTabInfos.Intersect 470
TRVTabInfos.Items 469
TRVTabItemInfo 526
TRVTableBorderStyle 645
TRVTableCellData 567, 608
TRVTableCellData.BackgroundImage 567
TRVTableCellData.BackgroundImageFileName 568
TRVTableCellData.BackgroundStyle 568
TRVTableCellData.BestHeight 568
TRVTableCellData.BestWidth 569
TRVTableCellData.BorderColor 569
TRVTableCellData.BorderLightColor 570
TRVTableCellData.Color 570
TRVTableCellData.ColSpan 570
TRVTableCellData.GetRVData 575
TRVTableCellData.Hint 571
TRVTableCellData.Opacity 571
TRVTableCellData.Options 571
TRVTableCellData.properties 576
TRVTableCellData.Rotation 573
TRVTableCellData.RowSpan 574
TRVTableCellData.Tag 574
TRVTableCellData.VAlign 575
TRVTableCellData.VisibleBorders 575
TRVTableCellOption 571
TRVTableCellOptions 571
TRVTableDrawBorderEvent 565
TRVTableItemInfo 526, 528, 562
SetRowKeepTogether 562
SetRowPageBreakBefore 562
SetRowVAlign 562
TRVTableItemInfo.AssignProperties 547
TRVTableItemInfo.BackgroundImage 530
TRVTableItemInfo.BackgroundImageFileName 531
TRVTableItemInfo.BackgroundStyle 531
TRVTableItemInfo.BestWidth 531
TRVTableItemInfo.BorderColor 532
TRVTableItemInfo.BorderHSpacing 532
TRVTableItemInfo.BorderLightColor 532
TRVTableItemInfo.BorderStyle 533
TRVTableItemInfo.BorderVSpacing 533
TRVTableItemInfo.BorderWidth 534
TRVTableItemInfo.CanMergeCells 547
TRVTableItemInfo.CanMergeSelectedCells 548
TRVTableItemInfo.CellBorderColor 534
TRVTableItemInfo.CellBorderLightColor 534
TRVTableItemInfo.CellBorderStyle 535
TRVTableItemInfo.CellBorderWidth 535
TRVTableItemInfo.CellHPadding 536
TRVTableItemInfo.CellHSpacing 536
TRVTableItemInfo.CellPadding 536
TRVTableItemInfo.Cells 537
TRVTableItemInfo.CellVPadding 537
TRVTableItemInfo.CellVSpacing 537
TRVTableItemInfo.Changed 548
TRVTableItemInfo.ColBandSize 538
TRVTableItemInfo.ColCount 538
TRVTableItemInfo.Color 538
TRVTableItemInfo.Create 548
TRVTableItemInfo.CreateEx 549
TRVTableItemInfo.DeleteCols 549
TRVTableItemInfo.DeleteEmptyCols 550
TRVTableItemInfo.DeleteEmptyRows 550
TRVTableItemInfo.DeleteRows 550
TRVTableItemInfo.DeleteSelectedCols 551
TRVTableItemInfo.DeleteSelectedRows 551
TRVTableItemInfo.Deselect 551
TRVTableItemInfo.Destroy 551
TRVTableItemInfo.EditCell 552
TRVTableItemInfo.EvenColumnsColor 546
TRVTableItemInfo.EvenRowsColor 546
TRVTableItemInfo.FirstColumnColor 546
TRVTableItemInfo.GetCellAt 552
TRVTableItemInfo.GetEditedCell 552
TRVTableItemInfo.GetNormalizedSelectionBounds 553
TRVTableItemInfo.GetSelectionBounds 553

TRVTableItemInfo.HeadingRowColor 546
TRVTableItemInfo.HeadingRowCount 539
TRVTableItemInfo.HOutermostRule 539
TRVTableItemInfo.HRuleColor 539
TRVTableItemInfo.HRuleWidth 540
TRVTableItemInfo.InsertCols 554
TRVTableItemInfo.InsertColsLeft 554
TRVTableItemInfo.InsertColsRight 554
TRVTableItemInfo.InsertRows 555
TRVTableItemInfo.InsertRowsAbove 555
TRVTableItemInfo.InsertRowsBelow 555
TRVTableItemInfo.IsCellSelected 556
TRVTableItemInfo.LastColumnColor 546
TRVTableItemInfo.LastRowColor 546
TRVTableItemInfo.LoadFromStream 556
TRVTableItemInfo.MergeCells 556
TRVTableItemInfo.MergeSelectedCells 557
TRVTableItemInfo.OddColumnsColor 546
TRVTableItemInfo.OddRowsColor 546
TRVTableItemInfo.OnCellEditing 564
TRVTableItemInfo.OnDrawBorder 565
TRVTableItemInfo.Opacity 540
TRVTableItemInfo.Options 540
TRVTableItemInfo.PrintOptions 542
TRVTableItemInfo.RowBandSize 538
TRVTableItemInfo.RowCount 543
TRVTableItemInfo.Rows 543
TRVTableItemInfo.SaveRowsToStream 557
TRVTableItemInfo.SaveToStream 557
TRVTableItemInfo.Select 558
TRVTableItemInfo.SelectCols 558
TRVTableItemInfo.SelectRows 558
TRVTableItemInfo.SetCellBackgroundImage 559
TRVTableItemInfo.SetCellBackgroundImageFileName 559
TRVTableItemInfo.SetCellBackgroundStyle 559
TRVTableItemInfo.SetCellBestHeight 559
TRVTableItemInfo.SetCellBestWidth 560
TRVTableItemInfo.SetCellBorderColor 560
TRVTableItemInfo.SetCellBorderLightColor 560
TRVTableItemInfo.SetCellColor 560
TRVTableItemInfo.SetCellHint 561
TRVTableItemInfo.SetCellOpacity 561
TRVTableItemInfo.SetCellOptions 561
TRVTableItemInfo.SetCellRotation 561
TRVTableItemInfo.SetCellTag 562
TRVTableItemInfo.SetCellVAlign 562
TRVTableItemInfo.SetCellVisibleBorders 562
TRVTableItemInfo.SetTableVisibleBorders 563
TRVTableItemInfo.SplitSelectedCellsHorizontally 563
TRVTableItemInfo.SplitSelectedCellsVertically 563
TRVTableItemInfo.TextColSeparator 543
TRVTableItemInfo.TextRowSeparator 544
TRVTableItemInfo.UnmergeCells 564
TRVTableItemInfo.UnmergeSelectedCells 564
TRVTableItemInfo.VisibleBorders 544
TRVTableItemInfo.VOutermostRule 545
TRVTableItemInfo.VRuleColor 545
TRVTableItemInfo.VRuleWidth 545
TRVTableOption 540
TRVTableOptions 540
TRVTablePrintOption 542
TRVTablePrintOptions 542
TRVTableRow 577
TRVTableRow.Items 577
TRVTableRow.KeepTogether 577
TRVTableRow.PageBreakBefore 578
TRVTableRow.VAlign 578
TRVTableRows 578
TRVTableRows.GetMainCell 579
TRVTableRows.Items 579
TRVTabNavigationType 157
TRVTag 645
TRVTextBackgroundKind 402
TRVTextBoxItemInfo 601
TRVTextBoxItemInfo.Create 602
TRVTextBoxItemInfo.CreateEx 602
TRVTextDrawState 645
TRVTextDrawStates 645
TRVTextEngine 403
TRVTextItemInfo 526
TRVTextOption 438
TRVTextOptions 438
TRVThumbnailMaker 615
TRVTInplaceRVData 608
TRVUnderlineType 442
TRVUndoType 647
TRVUnicodeString 647
TRVUnicodeTestResult 620
TRVUnits 648
TRVUnitsRect 615
TRVURLNeededEvent 266
TRVVAlign 648
TRVVerticalAnchor 599
TRVWriteHyperlink 267
TRVZoomMode 524
TwipsToUnits 406

TwipsToUnits 406
TRVStyle 406

- U -

UnderlineColor 442
TCustomRVFontInfo 442
UnderlineType 442
TCustomRVFontInfo 442
Undo 355
TRichViewEdit 355
Undo in RichViewEdit 121
Undo in TRichViewEdit 54
Undo/redo in Tables 121
UndoAction 355
TRichViewEdit 355
UndoLimit 295
TRichViewEdit 295
UndoName 355
TRichViewEdit 355
Unicode 64, 446
TFontInfo 446
Unicode in TRichView 64
UnicodeMode 274
TRVRTFReaderProperties 274
Units 280, 404, 487
TRVDocParameters 280
TRVPrint 487
TRVStyle 404
Units of measurement 74
UnitsPerInch 281
TRVDocParameters 281
UnmergeCells 564
TRVTableItemInfo 564
UnmergeSelectedCells 564
TRVTableItemInfo 564
UpdateModifiedParaStyleProperties 466
TRVStyleTemplate 466
UpdateModifiedTextStyleProperties 466
TRVStyleTemplate 466
UpdatePageNumbers 507
TRVReportHelper 507
UpdatePaletteInfo 238, 518
TCustomRVPrint 518
TRichView 238
URLFormatName 651
UseCharsetForUnicode 275
TRVRTFReaderProperties 275
UseHypertextStyles 275
TRVRTFReaderProperties 275
UseSingleCellPadding 275
TRVRTFReaderProperties 275
UseSound 404
TRVStyle 404
UseStyleTemplates 158
TRichView 158
UseXPThemes 511
TRVScroller 511

- V -

Valid documents 74
Valid TRichView documents 74
ValidParaProperties 463
TRVStyleTemplate 463
ValidTextProperties 464
TRVStyleTemplate 464
VAlign 159, 575, 578, 595
TRichView 159
TRVBoxProperties 595
TRVTableCellData 575
TRVTableRow 578
VerticalAlignment 600
TRVBoxPosition 600
VerticalAnchor 599
TRVBoxPosition 599
VerticalOffset 600
TRVBoxPosition 600
VerticalPositionKind 600
TRVBoxPosition 600
Viewer vs Editor 71
VisibleBorders 468, 544, 575
TRVBorder 468
TRVTableCellData 575
TRVTableItemInfo 544
VOutermostRule 545
TRVTableItemInfo 545
VRuleColor 545
TRVTableItemInfo 545
VRuleWidth 545
TRVTableItemInfo 545
VScrollMax 512
TRVScroller 512
VScrollPos 512
TRVScroller 512
VScrollVisible 512
TRVScroller 512
VShift 443
TCustomRVFontInfo 443
VSmallStep 159

VSmallStep 159
TRichView 159

- W -

WheelStep 512
TRVScroller 512
Width 468, 595
TRVBorder 468
TRVBoxProperties 595
WidthType 595
TRVBoxProperties 595
WordWrap 159
TRichView 159

- Z -

ZoomInCursor 523
TCustomRVPrintPreview 523
ZoomMode 281, 524
TCustomRVPrintPreview 524
TRVDocParameters 281
ZoomOutCursor 524
TCustomRVPrintPreview 524
ZoomPercent 281, 525
TCustomRVPrintPreview 525
TRVDocParameters 281