ooDialog Reference Manual

ooDialog an ooRexx™ Extension

Version 3.2.0 Edition
Revision 1
August 2012

W. David Ashley
Rony G. Flatscher
Mark Hessling
Rick McGuire
Mark Miesfeld
Lee Peedin
Oliver Sims
Jon Wolfers

ooDialog Reference Manual: ooDialog an ooRexx™ Extension

by
W. David Ashley
Rony G. Flatscher
Mark Hessling
Rick McGuire
Mark Miesfeld
Lee Peedin
Oliver Sims
Jon Wolfers

Version 3.2.0 Edition
Published August 2012
Copyright © 1995, 2004 IBM Corporation and others. All rights reserved.
Copyright © 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012 Rexx Language Association. All rights reserved.

This program and the accompanying materials are made available under the terms of the *Common Public License Version 1.0*.

Before using this information and the product it supports, be sure to read the general information under *Notices*.

This document was originally owned and copyrighted by IBM Corporation 1995, 2004. It was donated as open source under the *Common Public License Version 1.0* to the Rexx Language Association in 2004.

Thanks to Julian Choy for the ooRexx logo design.

Table of Contents

1. About This Book 1

1.1. Who Should Use This Book 1
1.2. How This Book is Structured 1
1.3. Related Information 1
1.4. How to Read the Syntax Diagrams 1
1.5. A Note About Program Examples in this Document 3
1.6. Getting Help 3
1.6.1. The Rexx Language Association Mailing List 3
1.6.2. The Open Object Rexx SourceForge Site 3
1.6.3. comp.lang.rexx Newsgroup 5

2. OODialog Method Reference 7

3. Definition of Terms 11

4. BaseDialog Class 15

4.1. Preparing and Running the Dialog 23
4.1.1. Init 24
4.1.2. InitAutoDetection 25
4.1.3. NoAutoDetection 26
4.1.4. AutoDetection 26
4.1.5. InitDialog 26
4.1.6. Run 27
4.1.7. Execute 27
4.1.8. ExecuteAsync 28
4.1.9. EndAsyncExecution 29
4.1.10. Popup 30
4.1.11. PopupAsChild 31
4.1.12. IsDialogActive 32
4.1.13. StopIt 33
4.1.14. HandleMessages 33
4.1.15. AsyncMessageHandling 33
4.1.16. PeekDialogMessage 33
4.1.17. ClearMessages 34
4.1.18. SendMessageToItem 34
4.2. Connect Event Methods 35
4.2.1. ConnectResize 35
4.2.2. ConnectMove 37
4.2.3. ConnectPosChanged 37
4.2.4. ConnectMouseCapture 38
4.2.5. ConnectHelp 39
4.2.6. ConnectKeyPress 40
4.2.7. ConnectFKeyPress 45
4.2.8. DisconnectKeyPress 47
4.2.9. HasKeyPressConnection 49
4.2.10. ConnectButton 50
4.2.11. ConnectBitmapButton 51
4.2.12. ConnectControl 53

4.2.13. ConnectDraw 54
4.2.14. ConnectList 55
4.2.15. ConnectListLeftDoubleClick 55
4.2.16. ConnectScrollBar 56
4.2.17. ConnectAllSBEvents 58
4.2.18. AddUserMsg 59
4.3. Connect Attribute Methods 61
4.3.1. ConnectEntryLine 61
4.3.2. ConnectComboBox 62
4.3.3. ConnectCheckBox 63
4.3.4. ConnectRadioButton 63
4.3.5. ConnectListBox 64
4.3.6. ConnectMultiListBox 64
4.3.7. AddAttribute 65
4.4. Get and Set Methods 66
4.4.1. GetData 66
4.4.2. SetData 67
4.4.3. ItemTitle 67
4.4.4. SetStaticText 68
4.4.5. GetEntryLine 68
4.4.6. SetEntryLine 68
4.4.7. GetListLine 69
4.4.8. SetListLine 69
4.4.9. GetMultiList 70
4.4.10. SetMultiList 70
4.4.11. GetComboLine 71
4.4.12. SetComboLine 71
4.4.13. GetRadioButton 72
4.4.14. SetRadioButton 72
4.4.15. GetCheckBox 72
4.4.16. SetCheckBox 73
4.4.17. GetValue 73
4.4.18. SetValue 74
4.4.19. GetAttrib 74
4.4.20. SetAttrib 75
4.4.21. SetDataStem 75
4.4.22. GetDataStem 76
4.5. Standard Event Methods 76
4.5.1. OK 76
4.5.2. Cancel 77
4.5.3. Help 77
4.5.4. Validate 77
4.5.5. Leaving 78
4.5.6. DeInstall 78
4.6. Combo Box Methods 78
4.6.1. AddComboEntry 78
4.6.2. InsertComboEntry 79
4.6.3. DeleteComboEntry 79

4.6.4. FindComboEntry 80
4.6.5. GetComboEntry 81
4.6.6. GetComboItems 81
4.6.7. GetCurrentComboIndex 81
4.6.8. SetCurrentComboIndex 82
4.6.9. ChangeComboEntry 83
4.6.10. ComboAddDirectory 83
4.6.11. ComboDrop 84
4.7. List Box Methods 85
4.7.1. GetListWidth 85
4.7.2. SetListWidth 85
4.7.3. SetListColumnWidth 86
4.7.4. AddListEntry 86
4.7.5. InsertListEntry 87
4.7.6. DeleteListEntry 87
4.7.7. FindListEntry 88
4.7.8. GetListEntry 88
4.7.9. GetListItems 89
4.7.10. GetListItemHeight 89
4.7.11. SetListItemHeight 90
4.7.12. GetCurrentListIndex 90
4.7.13. SetCurrentListIndex 90
4.7.14. ChangeListEntry 91
4.7.15. SetListTabulators 91
4.7.16. ListAddDirectory 92
4.7.17. ListDrop 92
4.8. Scroll Bar Methods 93
4.8.1. GetSBRange 93
4.8.2. SetSBRange 93
4.8.3. GetSBPos 94
4.8.4. SetSBPos 94
4.8.5. CombineELwithSB 95
4.8.6. DetermineSBPosition 96
4.9. Methods to Query Operating System Values 97
4.9.1. GetSelf 97
4.9.2. Get 98
4.9.3. GetItem 98
4.9.4. GetControlID 99
4.9.5. GetPos 101
4.9.6. GetButtonRect 101
4.9.7. GetWindowRect 102
4.9.8. GetSystemMetrics 102
4.10. Appearance and Behavior Methods 103
4.10.1. BackgroundColor 104
4.10.2. Show 104
4.10.3. ToTheTop 105
4.10.4. EnsureVisible 106
4.10.5. Minimize 107

4.10.6. Maximize 108
4.10.7. Restore 108
4.10.8. IsMinimized 109
4.10.9. IsMaximized 110
4.10.10. FocusItem 110
4.10.11. TabToNext 111
4.10.12. TabToPrevious 111
4.10.13. SetGroup 112
4.10.14. SetTabStop 113
4.10.15. EnableItem 114
4.10.16. DisableItem 114
4.10.17. HideItem 114
4.10.18. HideItemFast 115
4.10.19. ShowItem 115
4.10.20. ShowItemFast 115
4.10.21. HideWindow 116
4.10.22. HideWindowFast 116
4.10.23. ShowWindow 116
4.10.24. ShowWindowFast 117
4.10.25. SetWindowRect 117
4.10.26. RedrawWindow 118
4.10.27. ResizeItem 119
4.10.28. MoveItem 120
4.10.29. Center 120
4.10.30. SetWindowTitle 121
4.10.31. FileViewer Example Program 121
4.11. Window Draw Methods 125
4.11.1. DrawButton 125
4.11.2. RedrawRect 125
4.11.3. RedrawButton 126
4.11.4. RedrawWindowRect 127
4.11.5. ClearRect 127
4.11.6. ClearButtonRect 128
4.11.7. ClearWindowRect 128
4.12. Bitmap Methods 129
4.12.1. ChangeBitmapButton 129
4.12.2. GetBitmapSizeX 129
4.12.3. GetBitmapSizeY 129
4.12.4. DrawBitmap 130
4.12.5. ScrollBitmapFromTo 131
4.12.6. TiledBackgroundBitmap 131
4.12.7. BackgroundBitmap 132
4.12.8. DisplaceBitmap 132
4.12.9. GetBmpDisplacement 133
4.13. Device Context Methods 133
4.13.1. GetWindowDC 133
4.13.2. GetButtonDC 134
4.13.3. FreeWindowDC 134

4.13.4. FreeButtonDC 135
4.14. Text Methods 135
4.14.1. Write 135
4.14.2. ScrollText 136
4.14.3. ScrollInButton 137
4.14.4. ScrollButton 138
4.14.5. SetItemFont 138
4.15. Animated Buttons 139
4.15.1. AddAutoStartMethod 139
4.15.2. ConnectAnimatedButton 140
4.16. Menu Methods 142
4.16.1. ConnectMenuItem 142
4.16.2. EnableMenuItem 143
4.16.3. DisableMenuItem 143
4.16.4. CheckMenuItem 143
4.16.5. UncheckMenuItem 144
4.16.6. GrayMenuItem 144
4.16.7. SetMenuItemRadio 144
4.16.8. GetMenuItemState 145
4.17. Debugging Method 145
4.17.1. Dump 145
5. DialogControl Class 147
5.1. Appearance and Behavior Methods 150
5.1.1. Show 150
5.1.2. Hide 151
5.1.3. HideFast 151
5.1.4. ShowFast 151
5.1.5. Display 152
5.1.6. Enable 152
5.1.7. Disable 153
5.1.8. Group 153
5.1.9. TabStop 154
5.1.10. Resize 155
5.1.11. Move 156
5.1.12. SetForegroundWindow 157
5.1.13. ForegroundWindow 158
5.1.14. GetID 158
5.1.15. GetStyleRaw 158
5.1.16. GetRect 160
5.1.17. SetRect 160
5.1.18. GetClientRect 161
5.1.19. GetPos 162
5.1.20. GetSize 162
5.1.21. AssignFocus 162
5.1.22. GetFocus 163
5.1.23. SetFocus 163
5.1.24. SetFocusToWindow 163

5.1.25. SetColor 164
5.1.26. Update 165
5.1.27. Title 165
5.1.28. Title= 165
5.1.29. SetTitle 166
5.2. Miscellaneous Dialog Control Methods 166
5.2.1. ProcessMessage 166
5.2.2. Value 167
5.2.3. Value= 167
5.2.4. AssignWindow 168
5.3. Connect Event Methods 169
5.3.1. ConnectKeyPress 169
5.3.2. ConnectFKeyPress 172
5.3.3. DisconnectKeyPress 174
5.3.4. HasKeyPressConnection 176
5.4. Draw Methods 177
5.4.1. Draw 177
5.4.2. Clear 177
5.4.3. ClearRect 178
5.4.4. Redraw 178
5.4.5. RedrawRect 179
5.4.6. RedrawClient 180
5.5. Conversion Methods 180
5.5.1. LogRect2AbsRect 180
5.5.2. AbsRect2LogRect 181
5.5.3. ScreenToClient 182
5.5.4. ClientToScreen 182
5.6. Scroll Methods 183
5.6.1. Scroll 183
5.6.2. HScrollPos 183
5.6.3. VScrollPos 184
5.6.4. SetHScrollPos 184
5.6.5. SetVScrollPos 184
5.7. Mouse and Cursor Methods 185
5.7.1. CursorPos 185
5.7.2. SetCursorPos 185
5.7.3. RestoreCursorShape 186
5.7.4. Cursor_Arrow 187
5.7.5. Cursor_AppStarting 187
5.7.6. Cursor_Cross 188
5.7.7. Cursor_No 188
5.7.8. Cursor_Wait 188
5.7.9. GetMouseCapture 189
5.7.10. CaptureMouse 189
5.7.11. ReleaseMouseCapture 189
5.7.12. IsMouseButtonDown 190
5.8. Bitmap Methods 190
5.8.1. LoadBitmap 191

5.8.2. RemoveBitmap 191
5.9. Device Context Methods 192
5.9.1. GetDC 192
5.9.2. FreeDC 192
5.10. Text Methods 193
5.10.1. Write 193
5.10.2. WriteDirect 195
5.10.3. TransparentText 195
5.10.4. OpaqueText 196
5.10.5. WriteToWindow 196
5.10.6. WriteToButton 198
5.10.7. GetTextSize 199
5.10.8. SetFont 199
5.10.9. CreateFont 200
5.10.10. DeleteFont 201
5.10.11. FontToDC 201
5.10.12. FontColor 202
5.11. Graphic Methods 202
5.11.1. CreateBrush 203
5.11.2. CreatePen 203
5.11.3. ObjectToDC 204
5.11.4. DeleteObject 205
5.12. Graphic Drawing Methods 205
5.12.1. Rectangle 206
5.12.2. DrawLine 206
5.12.3. DrawPixel 207
5.12.4. GetPixel 207
5.12.5. DrawArc 208
5.12.6. GetArcDirection 209
5.12.7. SetArcDirection 209
5.12.8. DrawPie 209
5.12.9. FillDrawing 210
5.12.10. DrawAngleArc 210
6. UserDialog Class 213
6.1. Init 216
6.2. InitAutoDetection 217
6.3. Create 217
6.4. CreateCenter 219
6.5. DefineDialog 219
6.6. Load 220
6.7. LoadFrame 221
6.8. LoadItems 222
6.9. Add... Methods 223
6.9.1. AddButton 225
6.9.2. AddBitmapButton 227
6.9.3. AddIcon 230
6.9.4. AddGroupBox 231

6.9.5. AddText 232
6.9.6. AddEntryLine 233
6.9.7. AddPasswordLine 236
6.9.8. AddListBox 237
6.9.9. AddComboBox 238
6.9.10. AddCheckBox 239
6.9.11. AddRadioButton 240
6.9.12. AddRadioGroup 241
6.9.13. AddCheckGroup 242
6.9.14. AddInput 243
6.9.15. AddInputGroup 244
6.9.16. AddComboInput 245
6.9.17. AddInputStem 247
6.9.18. AddCheckBoxStem 248
6.9.19. AddRadioStem 249
6.9.20. AddScrollBar 249
6.9.21. AddButtonGroup 250
6.10. Frames and Rectangles 251
6.10.1. AddWhiteRect 252
6.10.2. AddWhiteFrame 253
6.10.3. AddGrayRect 253
6.10.4. AddGreyFrame 253
6.10.5. AddBlackRect 253
6.10.6. AddBlackFrame 254
6.11. OK and Cancel Push Buttons 254
6.11.1. AddOkCancelRightBottom 254
6.11.2. AddOkCancelLeftBottom 254
6.11.3. AddOkCancelRightTop 255
6.11.4. AddOkCancelLeftTop 255
6.12. Dialog Control Methods 255
6.12.1. StartIt 255
6.12.2. StopIt 256
6.13. Menu Methods 256
6.13.1. CreateMenu 256
6.13.2. AddPopupMenu 257
6.13.3. AddMenuItem 257
6.13.4. AddMenuSeparator 258
6.13.5. SetMenu 258
6.13.6. LoadMenu 258
7. PlainUserDialog Class and PlainBaseDialog Class 261
8. Resource File Dialogs 267
8.1. ResDialog Class 267
8.1.1. Init 267
8.1.2. StartIt 268
8.1.3. SetMenu 269
8.2. RcDialog Class 269
8.2.1. Init 270

9. CategoryDialog Class 273
9.1. Setting Up the Dialog 275
9.1.1. Init 275
9.1.2. InitCategories 277
9.1.3. DefineDialog 279
9.1.4. CategoryPage 280
9.1.5. CreateCategoryDialog 280
9.1.6. InitDialog 280
9.1.7. GetSelectedPage 281
9.1.8. CurrentCategory 281
9.1.9. NextPage 281
9.1.10. PreviousPage 282
9.1.11. ChangePage 282
9.1.12. PageHasChanged 282
9.1.13. StartIt 283
9.2. Connect... Methods 283
9.3. Methods for Dialog Items 284
9.4. Get and Set Methods 285
9.4.1. SetCategoryStaticText 285
9.4.2. GetCategoryEntryLine 285
9.4.3. SetCategoryEntryLine 285
9.4.4. GetCategoryListLine 286
9.4.5. SetCategoryListLine 286
9.4.6. GetCategoryListWidth 286
9.4.7. SetCategoryListWidth 286
9.4.8. GetCategoryMultiList 286
9.4.9. SetCategoryMultiList 286
9.4.10. GetCategoryComboLine 287
9.4.11. SetCategoryComboLine 287
9.4.12. GetCategoryRadioButton 287
9.4.13. SetCategoryRadioButton 287
9.4.14. GetCategoryCheckBox 287
9.4.15. SetCategoryCheckBox 288
9.4.16. GetCategoryValue 288
9.4.17. SetCategoryValue 288
9.4.18. GetCategoryAttrib 288
9.4.19. SetCategoryAttrib 288
9.5. Combo Box Methods 288
9.5.1. AddCategoryComboEntry 289
9.5.2. InsertCategoryComboEntry 289
9.5.3. DeleteCategoryComboEntry 289
9.5.4. FindCategoryComboEntry 290
9.5.5. GetCategoryComboEntry 290
9.5.6. GetCategoryComboItems 290
9.5.7. GetCurrentCategoryComboIndex 290
9.5.8. SetCurrentCategoryComboIndex 290
9.5.9. ChangeCategoryComboEntry 291
9.5.10. CategoryComboAddDirectory 291

9.5.11. CategoryComboDrop 291
9.6. List Box Methods 291
9.6.1. AddCategoryListEntry 291
9.6.2. InsertCategoryListEntry 291
9.6.3. DeleteCategoryListEntry 292
9.6.4. FindCategoryListEntry 292
9.6.5. GetCategoryListEntry 292
9.6.6. GetCategoryListItems 292
9.6.7. GetCurrentCategoryListIndex 292
9.6.8. SetCurrentCategoryListIndex 292
9.6.9. ChangeCategoryListEntry 293
9.6.10. SetCategoryListTabulators 293
9.6.11. CategoryListAddDirectory 293
9.6.12. CategoryListDrop 293
9.7. Appearance Modification Methods 293
9.7.1. EnableCategoryItem 294
9.7.2. DisableCategoryItem 294
9.7.3. ShowCategoryItem 294
9.7.4. HideCategoryItem 294
9.7.5. SetCategoryItemFont 294
9.7.6. FocusCategoryItem 294
9.7.7. ResizeCategoryItem 295
9.7.8. MoveCategoryItem 295
9.7.9. SendMessageToCategoryItem 295
10. Standard Dialogs, External Functions, and Public Routines 297
10.1. DlgUtil Class 298
10.1.1. version (Class Method) 298
10.2. Standard Dialog Classes 299
10.2.1. TimedMessage Class 300
10.2.1.1. Init 300
10.2.1.2. DefineDialog 302
10.2.1.3. Execute 302
10.2.2. InputBox Class 303
10.2.2.1. Init 303
10.2.2.2. DefineDialog 304
10.2.2.3. AddLine 304
10.2.2.4. Execute 304
10.2.3. PasswordBox Class 304
10.2.3.1. AddLine 305
10.2.4. IntegerBox Class 305
10.2.4.1. Validate 305
10.2.5. MultiInputBox Class 305
10.2.5.1. Init 306
10.2.6. ListChoice Class 307
10.2.6.1. Init 307
10.2.7. MultiListChoice Class 308
10.2.8. CheckList Class 309

10.2.8.1. Init 309
10.2.9. SingleSelection Class 311
10.2.9.1. Init 311
10.3. External Functions 312
10.3.1. InfoMessage 312
10.3.2. ErrorMessage 313
10.3.3. YesNoMessage 313
10.3.4. GetScreenSize 313
10.3.5. GetSysMetrics 313
10.3.6. PlaySoundFile 314
10.3.7. PlaySoundFileInLoop 314
10.3.8. StopSoundFile 314
10.3.9. GetFileNameWindow 314
10.3.10. SleepMS 316
10.3.11. WinTimer 316
10.4. Public Routines 316
10.4.1. Play Routine 317
10.4.2. InfoDialog Routine 317
10.4.3. ErrorDialog Routine 318
10.4.4. AskDialog Routine 318
10.4.5. FileNameDialog Routine 319
10.4.6. FindWindow Routine 321
10.4.7. ScreenSize Routine 321
10.4.8. SystemMetrics Routine 322
10.4.9. MSSleep Routine 323
10.4.10. TimedMessage Routine 324
10.4.11. InputBox Routine 325
10.4.12. PasswordBox Routine 325
10.4.13. IntegerBox Routine 325
10.4.14. MultiInputBox Routine 325
10.4.15. ListChoice Routine 325
10.4.16. MultiListChoice Routine 326
10.4.17. CheckList Routine 326
10.4.18. SingleSelection Routine 328
11. DlgArea and DlgAreaU Classes 329
11.1. DlgArea Class 329
11.1.1. Init 329
11.1.2. B 330
11.1.3. Bottom 330
11.1.4. CX 330
11.1.5. CY 331
11.1.6. H 331
11.1.7. HR 331
11.1.8. L 331
11.1.9. Left 332
11.1.10. Margin 332
11.1.11. R 332

11.1.12. Right 332
11.1.13. T 332
11.1.14. Top 332
11.1.15. W 333
11.1.16. WR 333
11.1.17. X 333
11.1.18. Y 334
11.1.19. DlgArea Example 334
11.2. DlgAreaU Class 336
11.2.1. Init 336
11.2.2. CorrectionFactor 337
11.2.3. LastError 337
11.2.4. NoMove 337
11.2.5. NoResize 337
11.3. Creating Resizeable Dialogs 338
11.4. Possible Problems 339
11.5. Sample Code 339
12. AnimatedButton Class 341
13. MessageExtensions Class 345
13.1. ConnectCommonNotify 345
13.2. ConnectTreeNotify 347
13.3. DefTreeDragHandler 350
13.4. ConnectListNotify 352
13.5. DefListDragHandler 355
13.6. ConnectListViewNotify 357
13.7. ConnectButtonNotify 362
13.8. ConnectEditNotify 364
13.9. ConnectListBoxNotify 366
13.10. ConnectComboBoxNotify 368
13.11. ConnectScrollBarNotify 370
13.12. ConnectTabNotify 373
13.13. ConnectSliderNotify 375
14. AdvancedControls Class 379
14.1. GetStaticControl 380
14.2. GetEditControl 380
14.3. GetButtonControl 381
14.4. GetRadioControl 382
14.5. GetCheckControl 383
14.6. GetListBox 384
14.7. GetComboBox 385
14.8. GetScrollBar 386
14.9. GetTreeControl 387
14.10. GetListControl 388
14.11. GetProgressBar 389
14.12. GetSliderControl 390
14.13. GetTabControl 391
14.14. ConnectTreeControl 392

14.15. ConnectListControl 392
14.16. ConnectSliderControl 393
14.17. ConnectTabControl 393
14.18. AddTreeControl 393
14.19. AddListControl 395
14.20. AddProgressBar 399
14.21. AddSliderControl 400
14.22. AddTabControl 402

15. StaticControl Class 405

16. EditControl Class 407

16.1. Selected 408
16.2. Select 408
16.3. ScrollCommand 409
16.4. LineScroll 410
16.5. EnsureCaretVisibility 411
16.6. FirstVisibleLine 411
16.7. IsModified 411
16.8. SetModified 412
16.9. Lines 412
16.10. LineIndex 413
16.11. LineFromIndex 413
16.12. LineLength 414
16.13. GetLine 415
16.14. GetText 415
16.15. SetText 416
16.16. ReplaceSelText 417
16.17. SetLimit 418
16.18. PasswordChar= 418
16.19. PasswordChar 419
16.20. SetReadOnly 419
16.21. SetMargins 420
16.22. Margins 420
16.23. SetCue 421
16.24. ShowBalloon 422
16.25. HideBalloon 425
16.26. AddStyle 426
16.27. RemoveStyle 427
16.28. ReplaceStyle 428
16.29. GetStyle 430

17. ListControl Class 433

17.1. View Styles 436
17.2. Methods of the ListControl Class 436
17.2.1. ReplaceStyle 436
17.2.2. AddStyle 437
17.2.3. RemoveStyle 438
17.2.4. AddExtendedStyle 439
17.2.5. RemoveExtendedStyle 442

17.2.6. ReplaceExtendedStyle 444
17.2.7. GetExtendedStyle 445
17.2.8. GetExtendedStyleRaw 447
17.2.9. GetHoverTime 448
17.2.10. SetHoverTime 449
17.2.11. Check 449
17.2.12. Uncheck 450
17.2.13. CheckAll 452
17.2.14. UncheckAll 452
17.2.15. GetCheck 453
17.2.16. IsChecked 454
17.2.17. InsertColumn 455
17.2.18. DeleteColumn 456
17.2.19. ModifyColumn 457
17.2.20. ColumnInfo 458
17.2.21. ColumnWidth 459
17.2.22. SetColumnWidth 460
17.2.23. StringWidth 461
17.2.24. Insert 461
17.2.25. Modify 462
17.2.26. SetItemText 463
17.2.27. SetItemState 464
17.2.28. Add 465
17.2.29. AddRow 467
17.2.30. Delete 467
17.2.31. DeleteAll 468
17.2.32. Items 469
17.2.33. Last 469
17.2.34. Prepare4nItems 469
17.2.35. SelectedItems 470
17.2.36. ItemInfo 470
17.2.37. ItemText 471
17.2.38. ItemState 472
17.2.39. Select 473
17.2.40. Deselect 473
17.2.41. Selected 474
17.2.42. LastSelected 474
17.2.43. Focused 474
17.2.44. Focus 474
17.2.45. DropHighlighted 475
17.2.46. FirstVisible 475
17.2.47. NextSelected 476
17.2.48. PreviousSelected 476
17.2.49. Next 476
17.2.50. Previous 477
17.2.51. NextLeft 477
17.2.52. NextRight 478
17.2.53. SmallSpacing 478

17.2.54. Spacing 478
17.2.55. RedrawItems 478
17.2.56. UpdateItem 479
17.2.57. Update 480
17.2.58. EnsureVisible 480
17.2.59. SetSmallImages 481
17.2.60. SetImages 482
17.2.61. RemoveSmallImages 483
17.2.62. RemoveImages 483
17.2.63. Find 483
17.2.64. FindPartial 484
17.2.65. FindNearestXY 485
17.2.66. Arrange 485
17.2.67. SnapToGrid 486
17.2.68. AlignLeft 486
17.2.69. AlignTop 486
17.2.70. ItemPos 487
17.2.71. SetItemPos 487
17.2.72. Edit 488
17.2.73. EndEdit 489
17.2.74. SubclassEdit 489
17.2.75. RestoreEditClass 489
17.2.76. ItemsPerPage 489
17.2.77. Scroll 490
17.2.78. BkColor 490
17.2.79. BkColor= 491
17.2.80. TextColor 491
17.2.81. TextColor= 491
17.2.82. TextBkColor 492
17.2.83. TextBkColor= 492
17.3. Notification Messages 492
18. ButtonControl Class 495
18.1. State 495
18.2. State= 496
18.3. Style= 498
18.4. ChangeBitmap 499
18.5. DisplaceBitmap 501
18.6. GetBmpDisplacement 501
18.7. Scroll 501
18.8. ScrollText 502
18.9. GetBitmapSizeX 503
18.10. GetBitmapSizeY 504
18.11. DrawBitmap 504
18.12. DimBitmap 505
18.13. ScrollBitmapFromTo 505

19. RadioButton Class **507**
19.1. IsChecked 507
19.2. Check 508
19.3. Uncheck 508
19.4. Indeterminate 508
20. CheckBox Class **509**
21. ListBox Class **511**
21.1. Add 512
21.2. Insert 512
21.3. Delete 513
21.4. DeleteAll 513
21.5. Find 513
21.6. SelectedIndex 514
21.7. Selected 514
21.8. SelectIndex 514
21.9. DeSelectIndex 515
21.10. Select 515
21.11. SelectRange 516
21.12. DeselectRange 516
21.13. Items 517
21.14. SelectedItems 517
21.15. SelectedIndexes 518
21.16. MakeFirstVisible 518
21.17. GetFirstVisible 519
21.18. GetText 519
21.19. Modify 519
21.20. SetTabulators 520
21.21. AddDirectory 521
21.22. SetWidth 522
21.23. Width 522
21.24. ItemHeight 523
21.25. ItemHeight= 523
21.26. ColumnWidth= 523
22. ComboBox Class **525**
22.1. Add 525
22.2. Insert 526
22.3. Delete 526
22.4. DeleteAll 527
22.5. Find 527
22.6. SelectedIndex 528
22.7. Selected 528
22.8. SelectIndex 528
22.9. Select 529
22.10. Items 529
22.11. GetText 530
22.12. Modify 530
22.13. AddDirectory 530

22.14. OpenDropDown 532
22.15. CloseDropDown 532
22.16. IsDropDownOpen 532
22.17. EditSelection 532

23. ScrollBar Class 535

23.1. SetRange 535
23.2. Range 536
23.3. SetPos 536
23.4. Position 537
23.5. DeterminePosition 537

24. PropertySheet Class 539

24.1. Init 539

25. ProgressBar Class 541

25.1. Step 541
25.2. SetPos 542
25.3. SetStep 542
25.4. SetRange 543

26. SliderControl Class 545

26.1. Pos= 546
26.2. SetPos 546
26.3. Pos 547
26.4. InitRange 547
26.5. SetMin 548
26.6. SetMax 549
26.7. Range 550
26.8. ClearTicks 550
26.9. CountTicks 551
26.10. GetTick 551
26.11. SetTickAt 551
26.12. SetTickFrequency 552
26.13. GetLineStep 553
26.14. GetPageStep 553
26.15. SetLineStep 554
26.16. SetPageStep 554
26.17. InitSelRange 555
26.18. SetSelStart 556
26.19. SetSelEnd 556
26.20. ClearSelRange 557
26.21. SelRange 558

27. TabControl Class 559

27.1. Insert 560
27.2. Modify 561
27.3. AddSequence 562
27.4. AddFullSeq 562
27.5. Items 563
27.6. Rows 563

27.7. ItemInfo 564
27.8. Delete 565
27.9. DeleteAll 565
27.10. Last 566
27.11. Selected 566
27.12. SelectedIndex 566
27.13. Select 566
27.14. SelectIndex 567
27.15. Focus 567
27.16. Focused 568
27.17. SetImages 568
27.18. RemoveImages 568
27.19. SetPadding 569
27.20. SetSize 569
27.21. PosRectangle 570
27.22. AdjustToRectangle 570
27.23. RequiredWindowSize 571

28. TreeControl Class 573

28.1. Methods of the TreeControl Class 574
28.1.1. Insert 574
28.1.2. Add 576
28.1.3. Modify 579
28.1.4. ItemInfo 581
28.1.5. Items 583
28.1.6. VisibleItems 583
28.1.7. Root 584
28.1.8. Parent 584
28.1.9. Child 585
28.1.10. Selected 585
28.1.11. DropHighlighted 586
28.1.12. FirstVisible 586
28.1.13. Next 586
28.1.14. NextVisible 587
28.1.15. Previous 587
28.1.16. PreviousVisible 588
28.1.17. Delete 588
28.1.18. DeleteAll 589
28.1.19. Collapse 589
28.1.20. CollapseAndReset 590
28.1.21. Expand 591
28.1.22. Toggle 591
28.1.23. EnsureVisible 592
28.1.24. Indent 593
28.1.25. Indent= 593
28.1.26. Edit 594
28.1.27. EndEdit 594
28.1.28. SubclassEdit 595

28.1.29. RestoreEditClass 595
28.1.30. Select 595
28.1.31. MakeFirstVisible 596
28.1.32. DropHighlight 596
28.1.33. SortChildren 597
28.1.34. SetImages 598
28.1.35. RemoveImages 599
28.1.36. HitTest 599
28.1.37. MoveItem 600
28.1.38. IsAncestor 601
28.2. Notification Messages 602
29. VirtualKeyCodes Class 603
29.1. Methods of the VirtualKeyCodes Class 603
29.1.1. VCode 603
29.1.2. KeyName 603
29.2. Symbolic Names for Virtual Keys 604
A. Notices 609
A.1. Trademarks 609
A.2. Source Code For This Document 610
B. Common Public License Version 1.0 611
B.1. Definitions 611
B.2. Grant of Rights 611
B.3. Requirements 612
B.4. Commercial Distribution 612
B.5. No Warranty 613
B.6. Disclaimer of Liability 613
B.7. General 614
Index 615

List of Tables

3-1. Symbolic IDs Used by ooDialog 11
3-2. ooDialog Supplied Icons 12
4-1. BaseDialog Instance Methods 17
5-1. DialogControl Instance Methods 148
6-1. UserDialog Instance Methods 215
7-1. PlainUserDialog and PlainBaseDialog Class Methods 262
10-1. Standard Dialogs, Functions, and Routines 297
11-1. DlgAreaU Automatic Resize Elements 338
11-2. DlgAreaU Non-Automatic Resize Elements 339
13-1. MessageExtensions Instance Methods 345
14-1. AdvancedControls Instance Methods 379
16-1. EditControl Instance Methods 407
17-1. ListControl Instance Methods 433
18-1. ButtonControl Instance Methods 495
19-1. RadioButton Instance Methods 507
21-1. ListBox Instance Methods 511
22-1. ComboBox Instance Methods 525
23-1. ScrollBar Instance Methods 535
25-1. ProgressBar Instance Methods 541
26-1. SliderControl Instance Methods 545
27-1. TabControl Instance Methods 559
28-1. TreeControl Instance Methods 573
29-1. Symbolic Names for Virtual Keys 604

Chapter 1. About This Book

This book describes the OODialog class library in Open Object Rexx on the Windows® platform, and how to use it to program user interfaces.

1.1. Who Should Use This Book

This book is intended for Object Rexx programmers who want to design graphical user interfaces for their applications.

1.2. How This Book is Structured

This document used to be divided into two parts - a tutorial and a reference. Unfortunately, the IBM Resource Workshop has not been contributed to the open source community and is therefore not a part of the Open Object Rexx project. So the tutorial section which covered the Resource Workshop has been eliminated from this document.

However, the reference material which describes the classes and methods in detail has been retained.

1.3. Related Information

Open Object Rexx: Programming Guide

Open Object Rexx: Reference

1.4. How to Read the Syntax Diagrams

Throughout this book, syntax is described using the structure defined below.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The `>>---` symbol indicates the beginning of a statement.

  The `--->` symbol indicates that the statement syntax is continued on the next line.

  The `>---` symbol indicates that a statement is continued from the previous line.

  The `---><` symbol indicates the end of a statement.

  Diagrams of syntactical units other than complete statements start with the `>---` symbol and end with the `--->` symbol.

- Required items appear on the horizontal line (the main path).

  ```
  >>-STATEMENT--required_item------------------------------------><
  ```

- Optional items appear below the main path.

```
>>-STATEMENT--+---------------+------------------------------><
              +-optional_item-+
```

- If you can choose from two or more items, they appear vertically, in a stack.

  If you must choose one of the items, one item of the stack appears on the main path.

```
>>-STATEMENT--+-required_choice1-+----------------------------><
              +-required_choice2-+
```

- If choosing one of the items is optional, the entire stack appears below the main path.

```
>>-STATEMENT--+------------------+----------------------------><
              +-optional_choice1-+
              +-optional_choice2-+
```

- If one of the items is the default, it appears above the main path and the remaining choices are shown below.

```
              +-default_choice--+
>>-STATEMENT--+-----------------+----------------------------><
              +-optional_choice-+
              +-optional_choice-+
```

- An arrow returning to the left above the main line indicates an item that can be repeated.

```
              +-----------------+
              V                 |
>>-STATEMENT----repeatable_item-+----------------------------><
```

  A repeat arrow above a stack indicates that you can repeat the items in the stack.

- A set of vertical bars around an item indicates that the item is a fragment, a part of the syntax diagram that appears in greater detail below the main diagram.

```
>>-STATEMENT--| fragment |-------------------------------------><

fragment:

|--expansion_provides_greater_detail---------------------------|
```

- Keywords appear in uppercase (for example, `PARM1`). They must be spelled exactly as shown but you can type them in upper, lower, or mixed case. Variables appear in all lowercase letters (for example, `parmx`). They represent user-supplied names or values.
- If punctuation marks, parentheses, arithmetic operators, or such symbols are shown, you must enter them as part of the syntax.

The following example shows how the syntax is described:

```
        +-,------+
        V        |
>>-MAX(----number-+--)----------------------------------------><
```

1.5. A Note About Program Examples in this Document

The program examples in this document are rendered in a mono-spaced font that is not completely compatible for cut-and-paste functionality. Pasteing text into an editor could result in some characters outside of the standard ASCII character set. Specifically, single-qoute and double-quote characters are sometimes converted incorrectly when pasted into an editor.

1.6. Getting Help

The Open Object Rexx Project has a number of methods to obtain help for ooRexx. These methods, in no particular order of preference, are listed below.

1.6.1. The Rexx Language Association Mailing List

The *Rexx Language Association* (http://www.rexxla.org/) maintains a mailing list for its members. This mailing list is only available to RexxLA members thus you will need to join RexxLA in order to get on the list. The dues for RexxLA membership are small and are charged on a yearly basis. For details on joining RexxLA please refer to the *RexxLA Home Page* (http://rexxla.org/) or the *RexxLA Membership Application* (http://rexxla.org/About_RexxLA/member.html) page.

1.6.2. The Open Object Rexx SourceForge Site

The Open Object Rexx Project (http://www.oorexx.org/) utilizes *SourceForge* (http://sourceforge.net/) to house the *ooRexx Project* (http://sourceforge.net/projects/oorexx) source repositories, mailing lists and other project features. Here is a list of some of the most useful facilities.

The ooRexx Forums

The ooRexx project maintains a set of forums that anyone my contribute to or monitor. They are located on the *ooRexx Forums* (http://sourceforge.net/forum/?group_id=119701) page. There are currently three forums available: Help, Developers and Open Discussion. In addition, you can monitor the forums via email.

The Developer Mailing List

You can subscribe to the oorexx-devel mailing list at *ooRexx Mailing List Subscriptions* (http://sourceforge.net/mail/?group_id=119701) page. This list is for discussing ooRexx project development activities. It also supports a historical archive of past messages.

The Users Mailing List

You can subscribe to the oorexx-users mailing list at *ooRexx Mailing List Subscriptions* (http://sourceforge.net/mail/?group_id=119701) page. This list is for discussing using ooRexx. It also supports a historical archive of past messages.

The Announcements Mailing List

You can subscribe to the oorexx-announce mailing list at *ooRexx Mailing List Subscriptions* (http://sourceforge.net/mail/?group_id=119701) page. This list is only used to announce significant ooRexx project events.

The Bug Mailing List

You can subscribe to the oorexx-bugs mailing list at *ooRexx Mailing List Subscriptions* (http://sourceforge.net/mail/?group_id=119701) page. This list is only used for monitoring changes to the ooRexx bug tracking system.

Support Requests

You can create a support request at *ooRexx Support Request* (http://sourceforge.net/tracker/?group_id=119701&atid=684731) page. Please be sure to log in to Sourceforge before creating the request so that it will record your e-mail address. This will allow SourceForge (and the ooRexx developers) a way to contact you when updates are made to your request. Otherwise you will need to manually check back on this page to track any updates to the request.

Also, please try to provide as much information in the support request as possible so that the developers can determine the problem as quickly as possible.

Bug Reports

You can create a bug report at *ooRexx Bug Report* (http://sourceforge.net/tracker/?group_id=119701&atid=684730) page. Please be sure to log in to Sourceforge before creating the report so that it will record your e-mail address. This will allow SourceForge (and the ooRexx developers) a way to contact you when updates are made to your report. Otherwise you will need to manually check back on this page to track any updates to the report.

Also, please try to provide as much information in the bug report as possible so that the developers can determine the problem as quickly as possible.

Patch Reports

If you create an enhancement patch for ooRexx please post the patch using the *ooRexx Patch Report* (http://sourceforge.net/tracker/?group_id=119701&atid=684732) page. Please be sure to log in to Sourceforge before creating the report so that it will record your e-mail address. This will allow SourceForge (and the ooRexx developers) a way to contact you when updates are made to your report. Otherwise you will need to manually check back on this page to track any updates to the report.

Also, please try to provide as much information in the patch report as possible so that the developers can evaluate the enhancement as quickly as possible.

Please do not post bug patches here, instead you should open a bug report and attach the patch to it.

1.6.3. comp.lang.rexx Newsgroup

The comp.lang.rexx (news:comp.lang.rexx) newsgroup is a good place to obtain help from many individuals within the Rexx community. You can obtain help on Open Object Rexx or on any number of other Rexx interpreters and tools.

Chapter 2. OODialog Method Reference

The classes provided by OODialog form a hierarchy as shown in The Hierarchy of OODialog Classes.

Figure 2-1. The Hierarchy of OODialog Classes

OODPlain.cls:

OODialog.cls:

The classes are:

PlainBaseDialog, BaseDialog

Base methods regardless of whether the dialog is implemented as a binary resource, a script, or dynamically. PlainBaseDialog provides limited functionality.

PlainUserDialog

Subclass of PlainBaseDialog used to create a dialog with all its control elements or to execute a dialog stored in a resource script (.RC). This class has limited functionality.

DynamicDialog, DialogExtensions, WindowBase, WindowExtensions

Internal mixin classes used to extend PlainBaseDialog, PlainUserDialog, BaseDialog, UserDialog, and DialogControl. The methods provided by these classes are not listed separately but are listed in BaseDialog or UserDialog.

UserDialog

Subclass of BaseDialog used to create a dialog with all its control elements, such as push buttons, check boxes, radio buttons, entry lines, and list boxes.

ResDialog

Subclass of BaseDialog for dialogs within a binary (compiled) resource file (.DLL).

CategoryDialog

Subclass of UserDialog used to create a dialog with several pages that overlay each other.

TimedMessage

Class to show a message window for a defined duration.

InputBox

Class to dynamically define a dialog with a message, one entry line, and two push buttons (OK, Cancel).

PasswordBox

Similar to InputBox, but keystrokes in the entry line are shown as asterisks (*).

IntegerBox

Similar to InputBox, but only numeric data can be entered in the entry line.

MultiInputBox

Similar to InputBox, but with multiple entry lines.

ListChoice

Class to dynamically define a dialog with a list box, where one line can be selected and returned to the caller.

MultiListChoice

Similar to ListChoice, but more than one line can be selected and returned to the caller.

CheckList

Class to dynamically define a dialog with a group of check boxes, which can be selected and returned to the caller.

SingleSelection

Class to dynamically define a dialog with a group of radio buttons, where one can be selected and returned.

Dialog

Subclass of UserDialog for simple dialogs. You can change the default dialog style from UserDialog to ResDialog.

AnimatedButton

Class to implement an animated button within a dialog.

DialogControl

Class to implement methods that are common to all dialogs and dialog controls.

TreeControl

Class to implement a tree to display the list of items in a dialog in a hierarchy.

ListControl

Class to implement a list view to display the items in a dialog as a collection.

ProgressBar

Class to implement a progress indicator within a dialog.

SliderControl

Class to implement a slider or trackbar within a dialog.

TabControl

Class to implement tabs, which can be compared to dividers in a notebook or labels in a file cabinet.

StaticControl

Class to query and modify static controls, such as static text, group boxes, and frames.

EditControl

Class to query and modify edit controls, which are also called entry lines.

ButtonControl

Class to implement push buttons within a dialog.

RadioButtonControl

Class to implement radio buttons within a dialog.

CheckBoxControl

Class to implement check boxes within a dialog.

ListBoxControl

Class to implement list boxes within a dialog.

ComboBoxControl

Class to implement a combo box, which combines a list box with an edit control.

ScrollBarControl

Class to implement a scroll bar within a dialog.

PropertySheetControl

Class to implement a property sheet, which is similar to a category dialog that spreads its dialog items over several pages (categories), where the individual pages are controlled by a tab control instead of radio buttons or combo box lists.

Chapter 3. Definition of Terms

resource id

The identification number of a dialog resource. There are several different types of dialog resources, menus, dialog controls, and bitmaps, to name a few. You assign IDs when you create the resource definition for your dialog. An ID can be either numerical (for example, 1) or symbolic (for example, "IDOK").

IDs must be unique for each resource of the same type. Although two resources of different types may have the same ID, when using symbolic IDs with ooDialog, it is advisable to give all resources unique numerical IDs.

symbolic id

Defining a symbolic name for each numeric resource ID is often useful in programs that work with resource IDs. The symbolic name is then used where ever a numeric resource ID is needed. Symbolic names are easier to remember than numeric IDs and can make the code easier to understand.

The mechanism ooDialog provides for using symbolic IDs is the ConstDir attribute of the dialog classes. This is a directory object where the indexes are symbolic IDs and the item at each index is the numerical value of the ID. The symbolic IDs in the following table are pre-defined by ooDialog and placed in the `ConstDir` when an instance of a dialog class is created.

Table 3-1. Symbolic IDs Used by ooDialog

Numeric ID	Symbolic ID	ResourceType
1	IDOK	Button Control
2	IDCANCEL	Button Control
9	IDHELP	Button Control
-1	IDC_STATIC	Static Control
1	IDI_DLG_OODIALOG	Icon
2	IDI_DLG_APPICON	Icon
3	IDI_DLG_APPICON2	Icon
4	IDI_DLG_OOREXX	Icon
IDI_DLG_DEFAULT	IDI_DLG_DEFAULT	Icon

#define statement

Define statements are often used in the C and C++ languages to define symbolic names for numerical values. Because of this, it is common in Windows programs with dialogs to define symbolic names for resource IDs. Most Windows resource editors use symbolic IDs, (some to a limited degree, others exclusively.) Often the define statements are put in a header file so they are available both to the resource compiler and to the program code. The defines take the form of: `#define symbolicName numericValue` as in this example:

```
#define   ID_PUSHBUTTON1  413
#define   ID_EDIT1        511
#define   ID_LISTBOX1     602
```

When ooDialog parses a resource script or a header file and finds a define statement, it will add the symbolic ID to the ConstDir directory object of the dialog. Resource scripts are used by subclasses of the `UserDialog` (see the Load method.) All the ooDialog dialog classes accept a header file as an optional parameter when a new instance of a dialog object is created. (See for example the `Init` method in the BaseDialog or UserDialog sections.) Symbolic IDs added to the `ConstDir` can be used in any method of the ooDialog classes where a resource ID is needed.

header file

A common practice when programming applications in Windows that use dialogs and dialog resources is to place symbolic defines in a separate file. These files often have a .h extension and are usually called header files. Windows resource editors often manage a header file for the symbolic IDs automatically. (For instance Microsoft's dialog editor creates, writes, and reads the resource ID header file completely on its own. The user does not need to take any action other than including the file in her program.)

handle

A unique reference to a Windows object assigned by the system. It can be a reference to a dialog, a particular dialog item, or a graphic object (pen, brush, font). Handles are required for certain methods; they can be retrieved from the system when needed.

dialog icon

The term *dialog icon* is used in this documentation to refer to the icon that is displayed in the left hand corner of the title bar of a dialog. In Windows this is often called the *application* icon. The dialog icon is also used for the Task Bar display and in the AltTab task switcher application.

The dialog icon for a specific dialog can be set when the dialog is run using one of the execute methods. See the Execute or Popup methods for example. ooDialog provides four pre-defined icons for use in dialogs. Custom icons can be used by including the icon in a binary (compiled) resource, a resource script, or by using the AddIcon method of the UserDialog. The following table shows the symbolic IDs of the pre-defined icons. Their numeric IDs should be considered reserved. There is a fifth symbolic ID that represents the default dialog icon. This ID can always be used where a dialog icon ID is needed.

Table 3-2. ooDialog Supplied Icons

Numeric ID	Symbolic ID
1	IDI_DLG_OODIALOG
2	IDI_DLG_APPICON
3	IDI_DLG_APPICON2
4	IDI_DLG_OOREXX
IDI_DLG_DEFAULT	IDI_DLG_DEFAULT

device context

Stores information about the graphic objects that are displayed, such as bitmaps, lines, and pixels, and the tools used to display them, such as pens, brushes, and fonts. A device context can be acquired for a dialog or a button; it must be explicitly freed when the text or graphic operations are completed.

pixel

Individual addressable point within a window. VGA screens support 640 by 480 pixels, SVGA screens support higher resolutions, such as 800 by 600, 1024 by 768, 1280 by 1024, and 1600 by 1200. Pixel values in a dialog start at the top left corner and include the window title and border.

dialog unit

Used within dialog box templates to define the size and position of the dialog box and its controls. There is a horizontal and a vertical dialog base unit to convert width and height of dialog boxes and controls from dialog units to pixels and vice versa. The value of these base units depend on the screen resolution and the active system font; they are stored in attributes of the UserDialog class.

```
xPixels = xDialogUnits * self~FactorX
```

color

Each color supported by the Windows operating system is assigned a number. Sample color indexes are 0 (black), 1 (dark red), 2 (dark green), 3 (dark yellow), 4 (dark blue), 5 (purple), 6 (blue grey), 7 (light grey), 8 (pale green), 9 (light blue), 10 (white), 11 (grey), 12 (dark grey), 13 (red), 14 (light green), 15 (yellow), 16 (blue), 17 (pink), 18 (turquoise).

color palette

An array that contains color values identifying the colors that can currently be displayed or drawn on the output device.

Color palettes are used by devices that can generate many colors but can only display or draw a subset of them at a time. For such devices, Windows maintains a system palette to track and manage the current colors of the device.

Applications do not have direct access to this system palette. Instead, Windows associates a default palette with each device context. Applications can use the colors in the default palette.

The default palette is an array of color values identifying the colors that can be used with a device context by default. Windows associates the default palette with a context whenever an application creates a context for a device that supports color palettes. The default palette ensures that colors are available for use by an application without any further action. The default palette typically has 20 entries (colors), but the exact number of entries can vary from device to device. The colors in the default palette depend on the device. Display devices, for example, often use the 16 standard colors of the VGA display and 4 other colors defined by Windows.

Windows documentation

The term *Windows documentation* is used throughout the ooDialog reference to refer to the Windows Operating System documentation provided by Microsoft. The documentation is called the **MSDN Library**. The library is provided online for anyone to access. In addition, since May 2006,

Microsoft has also provided free of charge the ISO images of the library installation program. Anyone can download the ISOs, burn them to a CD and install the library locally on their system.

It is not necessary for the ooDialog programmer to know or understand the underlying Windows API that ooDialog is built on. However, as programmers write more sophisticated ooDialog applications, it may prove helpful to look up certain details in the MSDN Library. The information below is provided to help the ooDialog programmer locate the MSDN Library, if they would like to. All things on the Internet change. The URLs listed here are accurate at the time of this writing.

The online MSDN Library is currently located at:

http://msdn2.microsoft.com/en-us/library/default.aspx.

Directions to the downloadable ISO images of the MSDN Library have been posted on this blog entry:

http://blogs.msdn.com/robcaron/archive/2006/07/26/678897.aspx

A Google search using: `"Rob Caron" General Downloads MSDN Library` should also turn up the blog entry.

Windows platform SDK

The *Windows Platform SDK* is provided free of charge by Microsoft. The SDK is not needed to write ooDialog programs. However, combining the use of the documentation in the MSDN Library with the SDK allows very sophisticated ooDialog programs to be written. In general, the ooDialog framework takes care of the low-level details needed to work with the Windows API. However, there are a few generic ooDialog methods that provide direct access to the Windows API.

As an example, the AddUserMessage method allows the programmer to connect any Windows message sent to a dialog to a method in his ooDialog class. To use this method, the programmer would go to the MSDN library to look up details on the message and message parameters he is interested in. He would then use the Platform SDK to determine the numeric value of the Windows message and possibly the numeric values of its parameters.

This link provides some good information on the Platform SDK in general and also points the reader to where to get a SDK.

http://en.wikipedia.org/wiki/Platform_SDK

Again, note that it is not at all necessary to obtain, or understand details concerning, the Platform SDK. This information is provided for those programmers that have reached the point where they think a method like `AddUserMessage` might help them and need some direction as to how to go about using it.

Chapter 4. BaseDialog Class

The BaseDialog class implements base methods for all dialogs regardless of whether the dialog is implemented as a binary resource, a resource script, or created dynamically. Binary (compiled) resources are stored in a DLL. A dialog is created dynamically by using Add... methods. Dialogs that are implemented using a resource script (.RC) are generated semi-dynamically.

BaseDialog is an abstract class. You cannot use it to execute a Windows dialog but have to use one of its subclasses.

See the subclasses UserDialog Class, ResDialog Class, and RcDialog Class for additional information.

Requires:

`BaseDlg.cls` is the source file of this class.

```
::requires oodialog.cls
```

Attributes:

Instances of the BaseDialog class have the following attributes:

AutoDetect

Automatic data field detection on (=1, default) or off (=0). For the UserDialog subclass the default is off and Connect... methods or a resource script are usually used.

AutomaticMethods

A queue containing the methods that are started concurrently before the execution of the dialog.

BkgBitmap

The handle to a bitmap that is displayed in the dialog's background.

BkgBrushBmp

The handle to a bitmap that is used to draw the dialog's background.

ConstDir

A directory object whose indexes are symbolic IDs. The entry for each index is the numerical value of the symbolic ID. Once a symbolic ID is added to the `Constdir` directory, the symbolic ID can be used in place of the numeric value in any method of the ooDialog classes that requires a resource ID.

A few symbolic IDs are pre-defined by ooDialog and are present in the `ConstDir` directory of all ooDialogs. See the table, Symbolic IDs Used by ooDialog, for a list of these IDs. The programmer can add symbolic IDs to this directory by using #define statements in a resource script or a header file. Of course, symbolic IDs can also be added directly in the program as the following code snippet shows:

```
::method Init
  forward class (super) continue
```

```
  self~ConstDir["ID_GB"] = 101
  self~ConstDir["ID_CB_REGINA"] = 107
  self~ConstDir["ID_CB_REGINALD"] = 111
  self~ConstDir["ID_CB_OOREXX"] = 115

...

::method defineDialog

  self~addGroupBox(10, 20, 150, 90, "Pick an interpreter", "BORDER", ID_GB)
  self~addCheckBox(ID_CB_REGINA, "cb1", 30, 40, , , "Regina", "GROUP")
  self~addCheckBox(ID_CB_REGINALD, "cb2", 30, 60, , , "Reginald")
  self~addCheckBox(ID_CB_OOREXX, "cb3", 30, 80, , , "ooRexx")

...

::method ok

  oorexxCB = self~getCheckControl(ID_CB_OOREXX)
  if oorexxCB~isChecked == "CHECKED" then
    say "You picked the right interpreter."
```

DataConnection

A protected attribute to store connections between dialog items and the attributes of the dialog instance.

DlgHandle

The handle to the dialog.

Finished

When this is set to 1, Ok or Cancel will terminate the dialog on exit.

InitCode

After the Init method has executed the InitCode attribute will be 0 if the object initialization detected no errors. The attribute will be non-zero if initialization failed or an error was detected. The programmer should always check the InitCode attribute after instantiating a dialog object. If the attribute is not zero, then the object was not initialized correctly and its behavior is undefined.

After the dialog is finished, the attribute will be 1 if terminated with Ok, 2 if terminated with Cancel.

IsExtended

A protected attribute that is true (=1) if the graphics extension is installed.

UseStem

A protected attribute that is true (=1) if a stem variable was passed to Init.

Routines:

See Public Routines for a description of the audio Play routine.

Methods:

Instances of the BaseDialog class implement the methods listed in Table 2.

Table 4-1. BaseDialog Instance Methods

Method...	...on page
AbsRect2LogRect	AbsRect2LogRect
AddAttribute	AddAttribute
AddAutoStartMethod	AddAutoStartMethod
AddComboEntry	AddComboEntry
AddListEntry	AddListEntry
AddUserMsg	AddUserMsg
AssignWindow	AssignWindow
AsyncMessageHandling	AsyncMessageHandling
AutoDetection	AutoDetection
BackgroundBitmap	BackgroundBitmap
BackgroundColor	BackgroundColor
Cancel	Cancel
CaptureMouse	CaptureMouse
Center	Center
ChangeBitmapButton	ChangeBitmapButton
ChangeComboEntry	ChangeComboEntry
ChangeListEntry	ChangeListEntry
CheckMenuItem	CheckMenuItem
Clear	Clear
ClearButtonRect	ClearButtonRect
ClearMessages	ClearMessages
ClearRect	ClearRect
ClearWindowRect	ClearWindowRect
ClientToScreen	ClientToScreen
CombineELwithSB	CombineELwithSB
ComboAddDirectory	ComboAddDirectory
ComboDrop	ComboDrop
ConnectAllSBEvents	ConnectAllSBEvents
ConnectAnimatedButton	ConnectAnimatedButton
ConnectBitmapButton	ConnectBitmapButton
ConnectButton	ConnectButton
ConnectCheckBox	ConnectCheckBox

Method...	...on page
ConnectComboBox	ConnectComboBox
ConnectControl	ConnectControl
ConnectDraw	ConnectDraw
ConnectEntryLine	ConnectEntryLine
ConnectFKeyPress	ConnectFKeyPress
ConnectHelp	ConnectHelp
ConnectKeyPress	ConnectKeyPress
ConnectList	ConnectList
ConnectListBox	ConnectListBox
ConnectListLeftDoubleClick	ConnectListLeftDoubleClick
ConnectMenuItem	ConnectMenuItem
ConnectMouseCapture	ConnectMouseCapture
ConnectMove	ConnectMove
ConnectMultiListBox	ConnectMultiListBox
ConnectPosChanged	ConnectPosChanged
ConnectRadioButton	ConnectRadioButton
ConnectResize	ConnectResize
ConnectScrollBar	ConnectScrollBar
CreateBrush	CreateBrush
CreateFont	CreateFont
CreatePen	CreatePen
Cursor_AppStarting	Cursor_AppStarting
Cursor_Arrow	Cursor_Arrow
Cursor_Cross	Cursor_Cross
Cursor_No	Cursor_No
CursorPos	CursorPos
Cursor_Wait	Cursor_Wait
DeInstall	DeInstall
DeleteComboEntry	DeleteComboEntry
DeleteFont	DeleteFont
DeleteListEntry	DeleteListEntry
DeleteObject	DeleteObject
DetermineSBPosition	DetermineSBPosition
Disable	Disable
DisableItem	DisableItem
DisableMenuItem	DisableMenuItem
DisconnectKeyPress	DisconnectKeyPress
DisplaceBitmap	DisplaceBitmap

Method...	...on page
Display	Display
Draw	Draw
DrawAngleArc	DrawAngleArc
DrawArc	DrawArc
DrawBitmap	DrawBitmap
DrawButton	DrawButton
DrawLine	DrawLine
DrawPie	DrawPie
DrawPixel	DrawPixel
Dump	Dump
Enable	Enable
EnableMenuItem	EnableMenuItem
EnableItem	EnableItem
EndAsyncExecution	EndAsyncExecution
EnsureVisible	EnsureVisible
Execute	Execute
ExecuteAsync	ExecuteAsync
FillDrawing	FillDrawing
FindComboEntry	FindComboEntry
FindListEntry	FindListEntry
FocusItem	FocusItem
FontColor	FontColor
FontToDC	FontToDC
ForegroundWindow	ForegroundWindow
FreeButtonDC	FreeButtonDC
FreeDC	FreeDC
FreeWindowDC	FreeWindowDC
Get	Get
GetArcDirection	GetArcDirection
GetAttrib	GetAttrib
GetBitmapSizeX	GetBitmapSizeX
GetBitmapSizeY	GetBitmapSizeY
GetBmpDisplacement	GetBmpDisplacement
GetButtonDC	GetButtonDC
GetButtonRect	GetButtonRect
GetClientRect	GetClientRect
GetCheckBox	GetCheckBox
GetComboEntry	GetComboEntry

Method...	...on page
GetComboItems	GetComboItems
GetComboLine	GetComboLine
GetControlID	GetControlID
GetCurrentComboIndex	GetCurrentComboIndex
GetCurrentListIndex	GetCurrentListIndex
GetData	GetData
GetDataStem	GetDataStem
GetDC	GetDC
GetEntryLine	GetEntryLine
GetFocus	GetFocus
GetID	GetID
GetItem	GetItem
GetListEntry	GetListEntry
GetListItemHeight	GetListItemHeight
GetListItems	GetListItems
GetListLine	GetListLine
GetListWidth	GetListWidth
GetMenuItemState	GetMenuItemState
GetMouseCapture	GetMouseCapture
GetMultiList	GetMultiList
GetPixel	GetPixel
GetPos	GetPos
GetRadioButton	GetRadioButton
GetRect	GetRect
GetSBPos	GetSBPos
GetSBRange	GetSBRange
GetSelf	GetSelf
GetSize	GetSize
GetSystemMetrics	GetSystemMetrics
GetTextSize	GetTextSize
GetValue	GetValue
GetWindowDC	GetWindowDC
GetWindowRect	GetWindowRect
GrayMenuItem	GrayMenuItem
HandleMessages	HandleMessages
HasKeyPressConnection	HasKeyPressConnection
HScrollPos	HScrollPos
Help	Help

Method...	...on page
Hide	Hide
HideFast	HideFast
HideItem	HideItem
HideItemFast	HideItemFast
HideWindow	HideWindow
HideWindowFast	HideWindowFast
Init	Init
InitAutoDetection	InitAutoDetection
InitDialog	InitDialog
IsMouseButtonDown	IsMouseButtonDown
Leaving	Leaving
InsertComboEntry	InsertComboEntry
InsertListEntry	InsertListEntry
IsDialogActive	IsDialogActive
IsMaximized	IsMaximized
IsMinimized	IsMinimized
IsDialogActive	IsDialogActive
ItemTitle	ItemTitle
ListAddDirectory	ListAddDirectory
ListDrop	ListDrop
LoadBitmap	LoadBitmap
LogRect2AbsRect	LogRect2AbsRect
Maximize	Maximize
Minimize	Minimize
Move	Move
MoveItem	MoveItem
NoAutoDetection	NoAutoDetection
ObjectToDC	ObjectToDC
OK	OK
OpaqueText	OpaqueText
PeekDialogMessage	PeekDialogMessage
Popup	Popup
PopupAsChild	PopupAsChild
Rectangle	Rectangle
Redraw	Redraw
RedrawClient	RedrawClient
RedrawButton	RedrawButton
RedrawRect	RedrawRect

Method...	...on page
RedrawWindow	RedrawWindow
RedrawWindowRect	RedrawWindowRect
ReleaseMouseCapture	ReleaseMouseCapture
RemoveBitmap	RemoveBitmap
Resize	Resize
ResizeItem	ResizeItem
Restore	Restore
RestoreCursorShape	RestoreCursorShape
Run	Run
ScreenToClient	ScreenToClient
Scroll	Scroll
ScrollBitmapFromTo	ScrollBitmapFromTo
ScrollButton	ScrollButton
ScrollInButton	ScrollInButton
ScrollText	ScrollText
SendMessageToItem	SendMessageToItem
SetArcDirection	SetArcDirection
SetAttrib	SetAttrib
SetCheckBox	SetCheckBox
SetComboLine	SetComboLine
SetCursorPos	SetCursorPos
SetCurrentComboIndex	SetCurrentComboIndex
SetCurrentListIndex	SetCurrentListIndex
SetData	SetData
SetDataStem	SetDataStem
SetEntryLine	SetEntryLine
SetFocus	SetFocus
SetFocusToWindow	SetFocusToWindow
SetFont	SetFont
SetForegroundWindow	SetForegroundWindow
SetGroup	SetGroup
SetHScrollPos	SetHScrollPos
SetVScrollPos	SetVScrollPos
SetItemFont	SetItemFont
SetListColumnWidth	SetListColumnWidth
SetListItemHeight	SetListItemHeight
SetListLine	SetListLine
SetListWidth	SetListWidth

Method...	...on page
SetListTabulators	SetListTabulators
SetMenuItemRadio	SetMenuItemRadio
SetMultiList	SetMultiList
SetRadioButton	SetRadioButton
SetRect	SetRect
SetSBPos	SetSBPos
SetSBRange	SetSBRange
SetStaticText	SetStaticText
SetTabStop	SetTabStop
SetTitle	SetTitle
SetValue	SetValue
SetWindowRect	SetWindowRect
SetWindowTitle	SetWindowTitle
Show	Show
ShowFast	ShowFast
ShowItem	ShowItem
ShowItemFast	ShowItemFast
ShowWindow	ShowWindow
ShowWindowFast	ShowWindowFast
StopIt	StopIt
TabToNext	TabToNext
TabToPrevious	TabToPrevious
TiledBackgroundBitmap	TiledBackgroundBitmap
Title	Title
Title=	Title=
TransparentText	TransparentText
ToTheTop	ToTheTop
UncheckMenuItem	UncheckMenuItem
Update	Update
Validate	Validate
VScrollPos	VScrollPos
Write	Write
WriteDirect	WriteDirect
WriteToButton	WriteToButton
WriteToWindow	WriteToWindow

4.1. Preparing and Running the Dialog

This section presents the methods used to prepare and initialize a dialog, show it, run it, and stop it.

4.1.1. Init

```
>>-aBaseDialog~Init(--+---------+--,--id--+-------------+--+----------+--)-->
                      +-Library-+         +-,--DlgData.-+  +-,--hFile-+
```

The constructor of the class installs the necessary C functions for the Object Rexx API manager and prepares the dialog management for a new dialog.

Protected:

This method is protected. You cannot create an instance of BaseDialog. You can only create instances of its subclasses.

Arguments:

The arguments are:

Library

The file name of a .DLL file. Pass an empty string if you do not use binary resources.

id

The resource ID, (numeric or symbolic,) of the dialog within the resource file.

DlgData.

A stem variable (remember the period!) that contains initialization data for the dialog. For example, if you assign the string "Hello world" to `DlgData.103`, where `103` is the ID of an entry field, it is initialized with this string. If the dialog is terminated with OK, the data of the dialog is copied into this stem variable.

hFile

A file, (often called a header file,) defining symbolic IDs for resources. The symbolic IDs defined within the file will be added to the ConstDir directory.

Example:

The following example shows how a header file, symbolic IDs, and the data stem can be used in instantiating a new object that is a subclass of the ResDialog. Assume `resources.h` is a file in the same directory as the program file and contains the following:

```
/* resources.h */
  #define IDD_BUILD_DLG           100
  #define IDC_EDITFIELD_INCLUDE   110
  #define IDC_COMBOBOX_PROJECT    120
  #define IDC_RADIO_DEBUG         130
  #define IDC_RADIO_RELEASE       131
```

```
#define IDC_CHECKBOX_CLEAN      140
#define IDI_DLG_ICON            514
```

The dialog (a fictitious build dialog) will have an entry line, a combo box, two radio buttons, and a check box. The `DlgData.` stem will be used to initialize the values of the controls using the symbolic IDs defined in the header file.

```
/* BuildDlg.rex */
  DlgData.IDC_EDITFIELD_INCLUDE = "C:\sdk\include"
  DlgData.IDC_COMBOBOX_PROJECT = "Calculator"
  DlgData.RADIO_DEBUG = 0
  DlgData.RADIO_RELEASE = 1
  DlgData.CHECKBOX_CLEAN = 0

  dlg = .BuildDialog~new( "dlg.dll", IDD_BUILD_DLG, DlgData, "resources.h" )
  if dlg~initCode <> 0 then do
    say "Error starting dialog.  initCode:" dlg~initCode
    return dlg~initCode
  end

  dlg~execute( "NORMAL", IDI_DLG_ICON )
  ...
```

At this point the dialog is shown, the entry line will contain "C:\sdk\include", the combo box will have the Calculator project selected, the release radio button will be checked, (the debug radio button will not be checked,) and the clean check box will not be checked.

The user interacts with the dialog and selects ok to close it. Now the state of the dialog when it was closed can be determined by checking the DlgData. stem values.

```
  ...
  dlg~deinstall

  if DlgData.CHECKBOX_CLEAN == 1 then doClean()

  includePath = DlgData.IDC_EDITFIELD_INCLUDE
  if DlgData.RADIO_RELEASE == 1 then
    success = doReleaseBuild(includePath)
  else
    success = doDebugBuild(includePath)

  return success
```

4.1.2. InitAutoDetection

```
>>-aBaseDialog~InitAutoDetection------------------------------><
```

The InitAutoDetection method is called by the Init method to change the default setting for the automatic data field detection.

Automatic data field detection means that for every dialog data item a corresponding Object Rexx attribute is created automatically. If you disable automatic detection, you have to use the Connect... methods to assign a dialog item to an Object Rexx attribute.

Protected:

This method is protected. You can override this method within your subclass to change the standard behavior.

Example:

The following example overrides the method to switch off auto detection:

```
::class MyDialog subclass UserDialog
::method InitAutoDetection
   self~NoAutoDetection
```

4.1.3. NoAutoDetection

```
>>-aBaseDialog~NoAutoDetection--------------------------------><
```

The NoAutoDetection method switches off auto detection.

4.1.4. AutoDetection

```
>>-aBaseDialog~AutoDetection----------------------------------><
```

The AutoDetection method switches on auto detection.

4.1.5. InitDialog

```
>>-aBaseDialog~InitDialog-------------------------------------><
```

The InitDialog method is called after the Windows dialog has been created. It is useful for setting data fields and initializing combo and list boxes. Do not use Set... methods because the SetData method is executed automatically afterwards and sets the values of all dialog items from the attributes.

Protected:

The method is designed to be overwritten in subclasses; it cannot be called from outside the class.

Example:

The following example shows how to use InitDialog to initialize dialog items; in this case a list box:

```
::class MyDialog subclass Userdialog
::method InitDialog
```

```
self~InitDialog:super
AddListEntry(501, "this is the first line")
AddListEntry(501, "and this one the second")
```

4.1.6. Run

```
>>-aBaseDialog~Run--------------------------------------------><
```

The Run method dispatches messages from the Windows dialog until the user terminates the dialog by one of the following actions:

- Press the OK button (the push button with ID 1)
- Press the Cancel button (the push button with ID 2)
- Press the Enter key (if OK or Cancel is the default button)
- Press the Esc key

Protected:

Run is a protected method. You cannot call this method directly; it is called by Execute.

4.1.7. Execute

```
                           +-DEFAULT--+
>>-aBaseDialog~Execute(--"--+----------+--"--+---------+--)----><
                           +-NORMAL---+     +-,--icon-+
                           +-SHOWTOP--+
                           +-HIDE-----+
                           +-MIN------+
                           +-MAX------+
                           +-INACTIVE-+
```

The Execute method creates the dialog, shows it (see Show), starts automatic methods, and destroys the dialog. The data is passed to the Windows dialog before execution and received from it after the dialog is terminated.

Note: If another dialog has already been started in the same process, it is disabled by Execute.

Arguments:

The arguments are:

show

See Show.

icon

The resource ID of the dialog's icon. If an icon ID is not supplied, the default ooDialog icon will be used.

Return value:

0

The dialog was not executed.

1

You terminated the method using the OK button.

2

You terminated the method using the Cancel button.

Example:

The following example instantiates a new dialog object (remember that it is not possible to instantiate an object of the BaseDialog class), creates a dialog template, and runs the dialog as the topmost window. The dialog icon is one of the pre-defined icons supplied by ooDialog:

```
MyDialog = .UserDialog~new(...)
MyDialog~Create(...)
MyDialog~Execute("SHOWTOP", IDI_DLG_OOREXX)
```

4.1.8. ExecuteAsync

```
>>-aBaseDialog~ExecuteAsync--+--------------+------------------->
                             +-(--sleeptime-+

>--+-----------------------------------------------+----------><
   +-+-----------------------------------------+--)-+
     +-,--+-------------------+--+---------+-+
          |    +-DEFAULT--+    |  +-,--icon-+
          +-"--+-NORMAL---+--"-+
               +-SHOWTOP--+
               +-HIDE-----+
               +-MIN------+
               +-MAX------+
               +-INACTIVE-+
```

The ExecuteAsync method does the same as Execute, except that it dispatches messages asynchronously. Therefore the ExecuteAsync method returns immediately after the dialog has been started.

Arguments:

The arguments are:

sleeptime

The time slice, in milliseconds, until the next message is processed.

show

See Show.

icon

The resource ID of the dialog's icon. If an icon ID is not supplied, the default ooDialog icon will be used.

Return value:

0

The dialog was started.

1

An error occurred. Do not call the EndAsyncExecution method in this case.

Example:

The following example starts a dialog and runs the statements between ExecuteAsync and EndAsyncExecution asynchronously to the dialog:

```
ret = MyDialog~ExecuteAsync(1000, "SHOWTOP")
if ret = 0 then do
   ...
   /* Object Rexx statements to run while the dialog is executing */
   ...
   MyDialog~EndAsyncExecution
   end
else call ErrorMessage("Could not start dialog")
```

4.1.9. EndAsyncExecution

```
>>-aBaseDialog~EndAsyncExecution------------------------------><
```

The EndAsyncExecution method is used to complete the asynchronous execution of a dialog. It does not terminate the dialog but waits until the user terminates it.

Return value:

0

The dialog was not executed.

1

The dialog was terminated using the OK button.

2

The dialog was terminated using the Cancel button.

Example:

See the example in ExecuteAsync.

4.1.10. Popup

```
                          +-DEFAULT--+
>>-aBaseDialog~Popup(--"--+----------+--"--+-------------------------------+--)-><
                          +-NORMAL---+     +-,--+-----------+--+---------+-+
                          +-SHOWTOP--+          +-sleeptime-+  +-,--icon-+
                          +-HIDE-----+
                          +-MIN------+
                          +-MAX------+
                          +-INACTIVE-+
```

The Popup method starts a dialog, dispatches messages asynchronously, and returns immediately after the dialog is started.

A dialog started with Popup is independent of any other dialog. This means that a dialog already started in the same process is not disabled by Popup. You can therefore use Popup to produce nonmodal dialogs.

Arguments:

The arguments are:

show

See Show.

sleeptime

The time, in milliseconds, until the next message is processed.

icon

The resource ID of the dialog's icon. If an icon ID is not supplied, the default ooDialog icon will be used.

Return value:

This method does not return a value.

Example:

The following example starts a dialog and runs the statements after Popup asynchronously to the dialog. This means that the dialog reacts to an event like pressing a button and calls the connected method while the DO loop is being processed:

```
MyDialog~Popup("SHOWTOP", 250)
do i = 1 to 1000
   say "Iteration" i
   call SleepMs 100
end
```

This example could also be part of a method handling an event of a dialog, for example dialog A. The newly started dialog MyDialog is independent of dialog A. If dialog A is closed, MyDialog remains unaffected and active.

4.1.11. PopupAsChild

```
                                          +-DEFAULT--+
>>-aBaseDialog~PopupAsChild(--parent--,--"--+----------+--"----->
                                          +-NORMAL---+
                                          +-SHOWTOP--+
                                          +-HIDE-----+
                                          +-MIN------+
                                          +-MAX------+
                                          +-INACTIVE-+

>--+-------------------------------+--)------------------------><
   +-,--+-----------+--+---------+-+
        +-sleeptime-+  +-,--icon-+
```

The PopupAsChild method starts a dialog as a child dialog of another dialog, dispatches messages asynchronously, and returns immediately after the dialog is started.

A dialog started with PopupAsChild and its parent dialog can be active at the same time. This means that the parent dialog is not disabled by the child dialog. You can therefore use PopupAsChild to produce nonmodal dialogs. However, the child dialog is automatically terminated when the parent dialog is closed.

Arguments:

The arguments are:

parent

An object of the PlainBaseDialog class or one of its descendants that is the parent of the newly started dialog.

show

See Show.

sleeptime

The time, in milliseconds, until the next message is processed.

icon

The resource ID of the dialog's icon. If an icon ID is not supplied, the default ooDialog icon will be used.

Return value:

This method does not return a value.

Example:

The following example starts a dialog and runs the statements after PopupAsChild asynchronously to the dialog. This means that the dialog reacts to an event like pressing a button and calls the connected method while the DO loop is being processed. The new dialog is started as a child of MyParent and is therefore closed when the MyParent dialog is closed:

```
MyParent = .UserDialog~new
...
MyParent~Popup("SHOWTOP")
...
MyDialog~PopupAsChild(MyParent, "SHOWTOP", 250)
do i = 1 to 1000
   say "Iteration" i
   call SleepMs 100
   if i = 800 then MyParent~Finished = 1  /* close both dialogs when i = 800 */
end
```

This example could also be part of a method handling an event of a dialog, for example dialog A. The newly started dialog MyDialog is independent of dialog A. If dialog A is closed, MyDialog remains unaffected and active.

4.1.12. IsDialogActive

```
>>-aBaseDialog~IsDialogActive---------------------------------><
```

The IsDialogActive method returns 1 if the Windows dialog still exists.

Example:

The following example tests whether the dialog is active:

```
if MyDialog~IsDialogActive then ...
```

4.1.13. StopIt

```
>>-aBaseDialog~StopIt----------------------------------------><
```

The StopIt method removes the Windows dialog from the memory. It is called by Execute, after the user terminates the dialog.

Protected:

This method is protected and for internal use only.

4.1.14. HandleMessages

```
>>-aBaseDialog~HandleMessages---------------------------------><
```

The HandleMessages method handles dialog messages synchronously. It is called by Execute. HandleMessages is a dispatcher that receives Windows events and posts the message that is set to handle the event.

4.1.15. AsyncMessageHandling

```
>>-aBaseDialog~AsyncMessageHandling(--sleeptime--)-------------><
```

The AsyncMessageHandling method starts the asynchronous handling of dialog messages. It is invoked automatically by ExecuteAsync with the Start method of the Object class. A message in this context is the name of an object method that is processed whenever the corresponding event occurs. You can set the messages that should be sent by using Connect... methods (see page Connect Methods).

Protected:

This method is protected and for internal use only.

Arguments:

The only argument is:

sleeptime

The time slice, in milliseconds, until the next message is processed.

4.1.16. PeekDialogMessage

```
>>-aBaseDialog~PeekDialogMessage------------------------------><
```

The PeekDialogMessage method returns the first pending message of the dialog's message queue without removing it from the message queue.

Return value:

The first pending message or an empty string.

4.1.17. ClearMessages

```
>>-aBaseDialog~ClearMessages----------------------------------><
```

The ClearMessages method clears all pending dialog messages.

4.1.18. SendMessageToItem

```
>>-aBaseDialog~SendMessageToItem(--id--,--msg--,--wp--,--lp--)-><
```

The SendMessageToItem method sends a Windows message to a dialog item. It is used to influence the behavior of dialog items.

Arguments:

The arguments are:

id

The ID of the dialog item.

msg

The Windows message (you need a Windows SDK to look up these numbers).

wp

The first message parameter (wParam).

lp

The second message parameter (lParam).

Example:

The following example sets the marker to radio button 9001:

```
MyDialog~SendMessageToItem(9001, "0x000000F1", 1, 0)
```

4.2. Connect Event Methods

The following methods create a connection between an event of the Windows dialog, or a dialog control, and a method of the ooRexx dialog object. In the Windows user interface, as the user interacts with the system, events are generated that specify what the action of the user was. Mouse clicks, keyboard presses, moving or sizing windows, all generate events. In the ooDialog framework, when a specific event is connected to a method, that method will be invoked each time the event occurs.

- For push buttons you connect a method to the button. The connected method is called each time the button is pressed (clicked.)
- List boxes, multiple list boxes, and combo boxes can also be connected to a method that is called each time a line in the box is selected.
- For a scroll bar you can specify different methods that are called depending on the user action. The user can click on the arrow buttons, drag the thumb, or use direction keys.

In a `UserDialog` the Connect... event methods for buttons are called automatically from the Add... methods. However, for most events like the resize or move events, the connection needs to be made explicitly. In general, events should be connected before the dialog is shown on the screen. So, the Connect... event methods can be used in the InitDialog method. In a UserDialog, the DefineDialog method is a good place for the Connect... event methods. If the programmer is overriding the Init method, the Connect... event methods can be placed there. But, be sure the methods are not used until after the superclass has been initialized.

> **Note:** The method name that is to be invoked when the specified event occurs must be less than 256 characters.

4.2.1. ConnectResize

```
>>-aBaseDialog~ConnectResize(--msgToRaise--)------------------><
```

The ConnectResize method connects a dialog resize event with a method. It is called each time the size of the dialog is changed.

Arguments:

The only argument is:

msgToRaise

The message that is to be sent each time the dialog is resized. Provide a method with a matching name.

Return value:

This method does not return a value.

Example 1:

```
::class MyDlgClass subclass UserDialog

::method Init
  self~init:super(...)
  self~ConnectResize("OnSize")

::method OnSize
  use arg sizeEvent, sizeinfo    /* sizeinfo contains information about the
                                    new width and height */
  msg = self~PeekDialogMessage
  if msg~left(6) = "ONSIZE" then return    /* not the last size
                                                        event message */
  w = binaryand(sizeinfo, "0x0000FFFF") / self~FactorX
  h = binaryand(sizeinfo, "0xFFFF0000") % X2D("FFFF") / self~FactorY
  say "New width=" w ", new height=" h
```

Example 2:

This example is pulled from the File Viewer example at the end of the "Appearance and Behavior Methods" section. A complete working example is presented there that uses a number of the base dialog methods.

```
::method defineDialog
  expose wasMinimized

  wasMinimized = .false
  style = "VSCROLL HSCROLL MULTILINE READONLY"
  self~addEntryLine(IDC_MULTILINE, "cEntry", 0, 0, 170, 180, style)
  self~connectResize("onSize")
  ...

/* The first arg, sizeEvent, is a flag that the OS sends specifying the type of
 * size event.  We are only interested in these 3 flags:
 *
 * SIZE_RESTORED   = 0
 * SIZE_MINIMIZED  = 1
 * SIZE_MAXIMIZED  = 2
 */
::method onSize
  expose wasMinimized
  use arg sizeEvent sizeInfo

  if sizeEvent = 1 then wasMinimized = .true

  if sizeEvent = 0 |  sizeEvent = 2 then do
    if \ wasMinimized then self~resizeEditControl
    wasMinimized = .false
  end
```

Note: Connections are usually placed in the Init or InitDialog method. If both methods are defined, use Init as the place for this connection - but not before init:super has been called.

4.2.2. ConnectMove

```
>>-aBaseDialog~ConnectMove(--msgToRaise--)--------------------><
```

The ConnectMove method connects a dialog move event with a method. It is called each time the position of the dialog is changed.

Arguments:

The only argument is:

msgToRaise

The message that is to be sent each time the dialog is moved. Provide a method with a matching name.

Return value:

This method does not return a value.

Example:

```
::class MyDlgClass subclass UserDialog

::method Init
  self~init:super(...)
  selfConnectMove("OnMove")

::method OnMove
  use arg dummy, posinfo     /* posinfo contains information
                                  about the new position */
  msg = self~PeekDialogMessage
  if msg~left(6) = "ONMOVE" then return    /* not the last move
                                                event message */
  w = binaryand(data, "0x0000FFFF") / self~FactorX
  h = binaryand(data, "0xFFFF0000") % X2D("FFFF") / self~FactorY
  say "New x=" x ", new y=" y
```

Note: Connections are usually placed in the Init or InitDialog method. If both methods are defined, use Init as the place for this connection - but not before init:super has been called.

4.2.3. ConnectPosChanged

```
>>-aBaseDialog~ConnectPosChanged(--msgToRaise--)--------------><
```

The ConnectPosChanged method connects a change regarding the dialog coordinates with a method. It is called each time the size, position, or place in the Z order of the dialog is changed.

Arguments:

The only argument is:

msgToRaise

The message that is to be sent each time the coordinates of the dialog are changed. Provide a method with a matching name.

Return value:

This method does not return a value.

Example:

```
::class MyDlgClass subclass UserDialog

::method Init
  self~init:super(...)
  self~ConnectPosChanged("OnNewPos")

::method OnNewPos
  say "The new rectangle is" self~GetWindowRect(self~DlgHandle)
```

Note: Connections are usually placed in the Init or InitDialog method. If both methods are defined, use Init as the place for this connection - but not before init:super has been called.

4.2.4. ConnectMouseCapture

```
>>-aBaseDialog~ConnectMouseCapture(--msgToRaise--)------------>>
```

The ConnectMouseCapture method connects a method with the lose-mouse-capture event. It is called each time the dialog loses the mouse capture. This can happen, for example, when you move a dialog with the mouse and release the left mouse button.

Arguments:

The only argument is:

msgToRaise

The message that is to be sent each time the mouse capture is lost in the dialog. Provide a method with a matching name.

Return value:

This method does not return a value.

4.2.5. ConnectHelp

```
>>-aBaseDialog~ConnectHelp(--msgToRaise--)----------------------><
```

The `ConnectHelp` method connects the Windows Help event with a method in the dialog class. The Windows Help event occurs when the user presses the F1 key. (Only the Help events generated when the dialog is the active window are connected.)

Arguments:

The only argument is:

msgToRaise

The name of the method that is to be invoked.

Return value:

0

No error.

1

An (internal) error prevented the message from being connected to a method.

Event Method Arguments

The ooDialog method connected to the Help event will receive the following four arguments in the order listed:

id

The resource ID of the dialog, dialog control, or menu item that had the focus when the F1 key was pressed.

type

Specifies if the ID in argument 1 was from a window (a dialog or dialog control) or from a menu item. This argument will either be `WINDOW` or `MENU`.

mouseX

The x coordinate of the mouse at the time the F1 key was pressed. This value is an absolute screen coordinate (pixel) and note that the mouse will not necessarily be over the dialog.

mouseY

The y coordinate of the mouse at the time the F1 key was pressed.

Example:

```
::method Init
    self~init:super
    ...
    self~connectResize(onResize)
    self~connectHelp(onHelp)
    ...

::method onHelp
    use arg id, type, mouseX, mouseY
    if type == "MENU" then w = 'Menu id' id; else w = 'Dialog id' id
    say "Help request:"
    say " " w
    say "  Mouse position x:" mouseX "y:" mouseY

/* As the user presses the F1 key at various times when the dialog has the focus
 * the output might be as follows:
 */

Help request:
  Dialog id 12
  Mouse position x: 420 y: 106
Help request:
  Menu id 60
  Mouse position x: 204 y: 93
Help request:
  Menu id 65
  Mouse position x: 203 y: 166
Help request:
  Dialog id 14
  Mouse position x: 218 y: 410
Help request:
  Dialog id 80
  Mouse position x: 387 y: 462
```

4.2.6. ConnectKeyPress

```
>>-aBaseDialog~ConnectKeyPress(--msgToRaise--,--keys-+------------+--)---->&lt;
                                                     +-,--filter--+
```

The `ConnectKeyPress` method connects a key press event with a method in the dialog class. A single key or multiple keys can be connected to the same method. Multiple methods can be connected for key press events, but only 1 method can be connected to any single key.

The underlying Windows dialog must exist before this method can be used. That means it can be used in InitDialog or any time thereafter. There is a maximum limit of 63 methods, per dialog, that can be connected to key press events. Connections can be removed (see the DisconnectKeyPress method) if there is no longer a need for a notification of a key press.

A similar method is a member of the `DialogControl` class. It is important to note this distinction between the two methods. The method of the `BaseDialog` (this method) will capture any key press event when the dialog is the active window. The method of the `DialogControl` class will only capture a key press when the control has the keyboard focus.

Arguments:

The arguments are:

msgToRaise

The name of the method that is to be invoked when the key press event happens.

keys

The key (or keys) for which the key press event is to be connected. A single key or multiple keys can be specified. A range of keys can be used. Each single key or range of keys is separated by a comma. A range of keyes is denoted by using the dash character "-". White space within the `keys` argument is ignored.

The keys are specified by the numeric value defined by Microsoft for its virtual key set. These numeric values are 0 through 255, although both 0 and 255 are not used. The VirtualKeyCodes class can be used to make a program more readable.

Note: The programmer can use the Windows documentation and Platform SDK to obtain the full list of the virtual key numbers.

In addition there are a few keywords that can be used to specify some common key ranges. These keywords are:

ALL

All keys.

FKEYS

All Function keys, other than F1. (In Windows the F1 key is the help key and the ConnectHelp method should be used for F1.)

ALPHA

The keys A though Z.

NUMERIC

The keys 0 through 9. Note that these are the normal number keys, not the keypad numbers on an enhanced keyboard.

ALPHANUMERIC

The keys A through Z and 0 through 9.

Note: Case is insignificant for these keywords as is the order of the keywords. A keyword not in the list will result in a return of -1. However, if the argument contains other valid key tokens, those keys will be connected to the method. If there are no other valid key tokens, then no connection is made.

filter

A (simplistic) filter that is applied to the key press event for the key(s) specified. The filter is a string of keywords separated by blanks. (Case is not significant, neither is the order of the words. Any words other than the specified keywords are ignored.) The possible keywords are: `SHIFT, CONTROL, ALT, AND, NONE.`

Shift, control, and alt specify that the corresponding key must be down at the time of the key press event. These keywords are combined in a boolean expression. The default is an OR expression. If the AND keyword is present then the boolean expression is an AND expression. If the NONE keyword is used, it means that none of the shift, control, or alt keys can be down at the time of the key press event. (When NONE is used, all other words in the string are ignored.)

Some examples may make this more clear:

```
::method initDialog

  -- Using the below, the onAltCD method would be invoked when the user types
  -- Alt-Shift-C or Alt-Shift-D.  But the method would not be invoked for Alt-C
  -- or Shift-D (or any other key press event.)

  keys = self~vCode('C') "," self~vCode('D')
  self~connectKeyPress(onAltCD, keys, "ALT AND SHIFT")

  -- The below would invoke the onAltCD method any time a C or a D was typed
  -- with either the Alt or the Control key down.  This would include Alt-C,
  -- Alt-Shift-C, Ctrl-Alt-Shift-C, etc..

  self~connectKeyPress(onAltCD, keys, "ALT CONTROL")

  -- The below would invoke the onAltCD method only when Alt-C or Alt-D was
```

```
-- typed.

self~connectKeyPress(onAltCD, keys, "ALT AND")

-- The below would invoke the onF4 method only when the F4 key was pressed by
-- itself. Alt-F4, Ctrl-F4, etc., would not invoke the method.

self~connectKeyPress(onF4, self~vCode('F4'), "NONE")
```

Return value:

The return values are:

0

Success.

-1

A problem with one of the arguments, such as skipping a required argument, using an incorrect format for the `keys` or the `filter` arguments, etc.. Note that it is possible to get a return of -1 but still have some keys connected. For instance in the following example the C and D keys would be connected and the filter applied. The ""dog"" token would result in -1 being returned:

```
keys = self~vCode('C') ", dog," self~vCode('D')
ret = self~connectKeyPress('onAltCD', keys, "ALT AND SHIFT")
say 'Got a return of:' ret
say "Have connection to onAltCD?" self~hasKeyPressConnection('onAltCD')

-- The output would be:
Got a return of: -1
Have connection to onAltCD? 1
```

-2

The underlying mechanism in the Windows API that is used to capture key events failed.

-4

An (internal) problem with the dialog window.

-5

Memory allocation error in the underlying Windows API.

-6

The maximum number of connections has been reached.

-7

The `msgToRaise` method is already connected to a key down event for this dialog.

Event Method Arguments

The ooDialog method connected to the key press event will receive the following five arguments in the order listed:

keyCode

The numeric code of the key pressed.

shift

A boolean (true or false) that denotes whether a shift key was down or up at the time of the key press. It will be true if a shift key was down and false if the shift key was not down.

control

True if a control key was down at the time of the key press, false if it was not.

alt

True if an alt key was down at the time of the key press, false if it was not.

extraInfo

This argument is a string containing keywords. It supplies extra information about the keyboard state at the time of a key press event. The string will contain some combination of these keywords

numOn

Num Lock was on at the time of the key press event.

numOff

Num Lock was off.

capsOn

Caps Lock was on at the time of the key press event.

capsOff

Caps Lock was off.

scrollOn

Scroll Lock was on at the time of the key press event.

scrollOff

Scroll Lock was off.

lShift

The left shift key was down at the time of the key press event.

rShift

The right shift key was down.

lControl

The left control key was down at the time of the key press event.

rControl

The right control key was down.

lAlt

The left alt key was down at the time of the key press event.

rAlt

The right alt key was down.

Example:

The following example is from a fictitious customer order system. As the user is filling out a customer order using the customer order dialog, he has the F2 through F5 short cut keys available. F2 brings up a customer look up dialog. F3 looks up info on the product number entered in an edit control. F4 resets the form by clearing all the fields. F5 is used to print out the finished invoice.

```
::method initDialog

  ...

  -- Capture F2 key presses, but not Ctrl-F2 or Alt-F2, etc..
  self~connectKeyPress(onF2, self~vCode('F2'), "NONE")

  -- Same idea for F3, F4, and F5.  This uses the actual numeric value for the
  -- keys without bothering to use the VirtualKeyCodes class to translate.
  self~connectKeyPress(onF3, 114, "NONE")
  self~connectKeyPress(onF4, 115, "NONE")
  self~connectKeyPress(onF5, 116, "NONE")

  ...

::method onF2
  self~showCustomerLookupDialog

::method onF3

  prodNum = self~getEditControl(IDC_EDIT_PRODUCT)~getText
  if prodNum \== "" then self~showProductInfo(prodNum)

::method onF4
  self~resetAllFields

::method onF5
  self~printInvoice
```

4.2.7. ConnectFKeyPress

```
>>-aDialogControl~ConnectFKeyPress(--msgToRaise--)------------><
```

The `ConnectFKeyPress` method connects a function key press to a method in the dialog instance. This works for function keys F2 through F24, without the shift, control, or alt keys also being pressed. The underlying Windows dialog must exist before this method can be invoked. This is a convenience method and is exactly equivalent to:

```
::method initDialog
  ...
  keys = self~vCode('F2') "-" self~vCode('F24')
  self~connectKeyPress(msgToRaise, keys, "NONE")
```

Arguments:

The required argument is:

msgToRaise

The method to be invoked for the F key press event.

Return value:

The return values are:

0

Success.

-2

The underlying mechanism in the Windows API that is used to capture key events failed.

-4

An (internal) problem with the dialog window.

-5

Memory allocation error in the underlying Windows API.

-6

The maximum number of connections has been reached.

-7

The `msgToRaise` method is already connected to a key down event for this dialog.

Example:

The following example is a variation on the example shown for the ConnectKeyPress method. It connects all the function keys to the same method and then determines what action to take by examining which key was pressed.

```
::method initDialog

  ...

  -- Capture all function key presses.
  self~connectFKeyPress(onFKey)

  ...

::method onFKey
  use arg keyPressed

  select
    when self~keyName(keyPressed) == 'F2' then self~showCustomerLookupDialog

    when keyPressed = 114 then do
      prodNum = self~getEditControl(IDC_EDIT_PRODUCT)~getText
      if prodNum \== "" then self~showProductInfo(prodNum)
    end

    when keyPressed = 115 then self~resetAllFields
    when keyPressed = 116 then self~printInvoice

    otherwise do
      -- Not interested in any other function keys
      nop
    end
  end
```

4.2.8. DisconnectKeyPress

```
>>-aBaseDialog~DisconnectKeyPress(--+---------------+--)-------><
                                    +--methodName--+
```

The `DisconnectKeyPress` method disconnects a key press event from a method that was previously connected using ConnectKeyPress. If no method name is specified than all key press event connections are removed. Otherwise, only the key press events connected to the specific method are removed.

Arguments:

The single optional argument is:

methodName

If `methodName` is specified, only the key press events connected to that method are disconnected. If the argument is omitted, then all key press events for the dialog will be disconnected.

Return value:

The return values are:

0

Success.

-1

The specified `methodName` is not connected to any key press events for this dialog.

-2

This return code can only occur when removing the connection for a single `methodName`. It indicates that after removing the connection, the underlying mechanism in the Windows API that captures the key press events could not be re-initialized. (It is doubtful this error would ever occur.)

Example:

The following example is a variation on the example shown for the ConnectKeyPress method. It builds on the fictitious customer order system. The F7 key saves the completed invoice into the system and enters a different phase of the companies business process. At this point (for whatever fictitious business reason) the fields can no longer be cleared and the user is not allowed to look up customer or product information. But, the user may still need to print the invoice. To prevent the accidental press of the hot keys causing the wrong action, those key presses are disconnected.

To demonstrate how key press connections can be added and removed through out the live time of the dialog, this example adds the F9 hot key. F9 starts a new order entry cycle and re-connects the hot keys used during the creation of a customer invoice. When the user then saves the next completed invoice, key press connections are removed, when she starts a new invoice key press connections are restored. This cycle could continue though out the day without the user ever closing the main dialog.

```
::method initDialog

  ...

  -- Capture F2 key presses, but not Ctrl-F2 or Alt-F2, etc..
  self~connectKeyPress(onF2, self~vCode('F2'), "NONE")

  -- Same idea for F3, F4, F5, and F7.  This uses the actual numeric value for
  -- the keys without bothering to use the VirtualKeyCodes class to translate.
```

```
  self~connectKeyPress(onF3, 114, "NONE")
  self~connectKeyPress(onF4, 115, "NONE")
  self~connectKeyPress(onF5, 116, "NONE")
  self~connectKeyPress(onF7, 118, "NONE")
  self~connectKeyPress(onF9, 120, "NONE")

  ...

::method onF2
  self~showCustomerLookupDialog

::method onF3

  prodNum = self~getEditControl(IDC_EDIT_PRODUCT)~getText
  if prodNum \== "" then self~showProductInfo(prodNum)

::method onF4
  self~resetAllFields

::method onF5
  self~printInvoice

::method onF7

  self~saveToDataBase
  self~disconnectKeyPress(onF2)
  self~disconnectKeyPress(onF3)
  self~disconnectKeyPress(onF4)

::method onF9

  self~resetAllFields
  self~connectKeyPress(onF2, 112, "NONE")
  self~connectKeyPress(onF3, 114, "NONE")
  self~connectKeyPress(onF4, 115, "NONE")
```

4.2.9. HasKeyPressConnection

```
>>-aBaseDialog~HasKeyPressConnection(--+--------------+--)-------><
                                       +--methodName--+
```

This method is used to query if a connection to a key press event already exists. It returns true or false so it can always be used in a boolean expression. If no argument is specified the method returns true if any key press events have been connected for the dialog and false otherwise. When the `methodName` argument is used, the query is only if there are key press events connected to the specified method.

Arguments:

The single optional argument is:

methodName

Query if any key press events are connected to this method.

Return value:

The returned value is always `true` or `false`.

true

A connection exists.

false

No connection exists.

Example:

The following example could come from a dialog where the user has the option to use hot keys or not. When the reset button is pushed the state of the dialog fields are reset. The hot keys enabled check box is set to reflect whether hot keys are currently enabled or not.

```
::method defineDialog

  ...
  self~addCheckBox(IDC_CHECK_FKEYSENABlED, , 30, 60, , , "Hot Keys Enabled")
  ...
  self~addButton(IDC_PB_RESET, 60, 135, 45, 15, "Reset", onReset)
  ...

::method onReset

  ...
  if self~hasKeyPressConnection then
    self~getCheckControl(IDC_CHECK_FKEYSENABlED)~check
  else
    self~getCheckControl(IDC_CHECK_FKEYSENABlED)~uncheck
  ...
```

4.2.10. ConnectButton

```
>>-aBaseDialog~ConnectButton(--id--+---------------+--)-------><
                                   +-,--msgToRaise-+
```

The ConnectButton method connects a push button with a method.

Arguments:

The arguments are:

id

The ID of the dialog element.

msgToRaise

The message that is sent each time the button is clicked. You should provide a method with the matching name.

Return value:

-1

The specified symbolic ID could not be resolved.

0

No error.

Example:

```
::class MyDlgClass subclass UserDialog

::method Init
  self~init:super(...)
  self~ConnectButton(203, "SayHello")

::method SayHello
  say "Hello"
```

Note: Connections are usually placed in the Init or InitDialog method. If both methods are defined, use Init as the place for this connection - but not before init:super has been called.

4.2.11. ConnectBitmapButton

```
>>-aBaseDialog~ConnectBitmapButton(--id--,--+------------+--,--bmpNormal-->
                                            +-msgToRaise-+

>--+-------------------------------------------------------------------------+--)-><
   +-,--bmpFocused--+-------------------------------------------------------+-+
                    +-,--bmpSelected--+-------------------------------------+-+
                                      +-,--bmpDisabled--+-----------------+-+
                                                        +-,--styleOptions-+
```

The ConnectBitmapButton method connects a bitmap and a method with a push button. The given bitmaps are displayed instead of a Windows push button.

Arguments:

The arguments are:

id

The ID of the button.

msgToRaise

The message that is to be sent to this object when the button is clicked.

bmpNormal

The name (alphanumeric), resource ID (numeric), or handle (*INMEMORY* option) of a bitmap file. This bitmap is displayed when the button is not selected, not focused, and not disabled. It is used for the other button states in case the other arguments are omitted.

bmpFocused

This bitmap is displayed when the button is focused. The focused button is activated when the Enter key is pressed.

bmpSelected

This bitmap is displayed while the button is clicked and held.

bmpDisabled

This bitmap is displayed when the button is disabled.

styleOptions

One of the following keywords:

FRAME

Draws a frame around the button. When using this option, the bitmap button behaves like a normal Windows button, except that a bitmap is shown instead of a text.

USEPAL

Stores the colors of the bitmap file as the system color palette. This option is needed when the bitmap was created with a palette other than the default Windows color palette. Use it for one button only, because only one color palette can be active at any time. *USEPAL* is invalid for a bitmap loaded from a DLL.

INMEMORY

This option must be used if the named bitmaps are already loaded into memory by using the LoadBitmap method. In this case, bmpNormal, bmpFocused, bmpSelected, and bmpDisabled specify a bitmap handle instead of a file.

STRETCH

If this option is specified and the extent of the bitmap is smaller than the extent of the button rectangle, the bitmap is adapted to match the extent of the button. STRETCH has no effect for bitmaps loaded through a DLL.

Return value:

-1

The specified symbolic ID could not be resolved.

0

No error.

Example:

The following example connects a button with four bitmaps and a method:

```
      .
      .
      .
::method InitDialog
  self~ConnectBitmapButton(204, "BmpButtonClicked", ,
                           "AddBut_n.bmp", "AddBut_f.bmp", ,
                           "AddBut_s.bmp", "AddBut_d.bmp", "FRAME")
::method BmpButtonClicked
      .
      .
      .
```

See also method ChangeBitmapButton.

4.2.12. ConnectControl

```
>>-aBaseDialog~ConnectControl(--id--+---------------+--)------><
                                    +-,--msgToRaise-+
```

The ConnectControl method connects a dialog control with a method.

Arguments:

The arguments are:

id

The ID of the dialog element.

msgToRaise

The message that is to be sent each time the button is clicked. Provide a method with the matching name.

Return value:

-1

The specified symbolic ID could not be resolved.

0

No error.

4.2.13. ConnectDraw

```
>>-aBaseDialog~ConnectDraw--(--id--+---------------+--)-------><
                                   +-,--msgToRaise-+
```

The ConnectDraw method connects the WM_DRAWITEM event with a method. A WM_DRAWITEM message is sent for owner-drawn buttons each time they are to be redrawn.

Arguments:

The arguments are:

id

The ID of the dialog control. If the ID is omitted, all drawing events of all owner-drawn buttons are routed to the method.

msgToRaise

The message that is to be sent each time the WM_DRAWITEM event occurs. Provide a method with the matching name. You can use `USE ARG ID` to retrieve the ID of the item that is to be redrawn.

Return value:

-1

The specified symbolic ID could not be resolved.

0

No error.

4.2.14. ConnectList

```
>>-aBaseDialog~ConnectList(--id--+---------------+--)---------->&lt;
                                 +-,--msgToRaise-+
```

The ConnectList method connects a list box, multiple list box, or combo box with a method. The method is called each time the user selects a new item from the list.

Arguments:

The arguments are:

id

The ID of the dialog element.

msgToRaise

The message that is to be sent each time the button is pressed. Provide a method with the matching name.

Return value:

-1

The specified symbolic ID could not be resolved.

0

No error.

4.2.15. ConnectListLeftDoubleClick

```
>>-aBaseDialog~ConnectListLeftDoubleClick(--id--,--msgToRaise--)-><
```

The ConnectListLeftDoubleClick method combines a left double-click within the list box with a method.

Arguments:

The arguments are:

id

The ID of the list box.

msgToRaise

The name of the method that is to be called.

Return value:

-1

The specified symbolic ID could not be resolved.

0

No error.

4.2.16. ConnectScrollBar

```
>>-aBaseDialog~ConnectScrollBar(--id--,--msgWhenUp--,--msgWhenDown-->

>--+-------------------------------------------------------------------+-->
   +-,--+-------------+--+-------------------------------------------+-+
        +-msgWhenDrag-+  +-,--+-----+--+---------------------------+-+
                              +-min-+  +-,--+-----+--+------------+-+
                                           +-max-+  +-,--+-----+-+
                                                        +-pos-+

>--+-------------------------------------------------------+-->
   +-,--+----------+--+-------------------------------------+-+
        +-progpgup-+  +-,--+----------+--+----------------+-+
                           +-progpgdn-+  +-,--+---------+-+
                                               +-progtop-+

>--+-------------------+--+-----------------------------------+-->
   +-,--+------------+-+  +-,--+-----------+--+--------------+-+
        +-progbuttom-+         +-progtrack-+  +-,--progendsc-+


>--)-------------------------------------------------------><
```

The ConnectScrollBar method initializes and connects a scroll bar to an Object Rexx object. Use this method in the InitDialog method.

Protected:

This method is protected.

Arguments:

The arguments are:

id

The ID of the scroll bar.

msgWhenUp

The method that is called each time the scroll bar is incremented.

msgWhenDown

The method that is called each time the scroll bar is decremented.

msgWhenDrag

The method that is called each time the scroll bar is dragged with the mouse.

min, max

The minimum and maximum values for the scroll bar.

pos

The current or preselected value.

progpgup

The method that is called each time the scroll bar is focused and the PgUp key is pressed.

progpgdn

The method that is called each time the scroll bar is focused and the PgDn key is pressed.

progtop

The method that is called each time the scroll bar is focused and the Home key is pressed.

progbottom

The method that is called each time the scroll bar is focused and the End key is pressed.

progtrack

The method that is called each time the scroll box is dragged.

progendsc

The method that is called each time the scroll box is released after the dragging.

Return value:

-1

The specified symbolic ID could not be resolved.

0

No error.

Example:

In the following example, scroll bar 255 is connected to three methods and initialized with 1 as the minimum, 20 as the maximum, and 6 as the current value:

```
::class MyDialog subclass UserDialog
      .
      .
      .
::method DefineDialog
self~ConnectScrollBar(255,"Increase","Decrease","Drag",1,20,6)
      .
      .
      .
::method Increase
      .
      .
      .
::method Decrease
      .
      .
      .
::method Drag
      .
      .
      .
/* see CombineElWithSB below for continuation */
```

4.2.17. ConnectAllSBEvents

```
>>-aBaseDialog~ConnectAllSBEvents(--id--,--Prog----------------->

>--+-----------------------------------------+--)--------------><
   +-,--+-----+--+-----------------------+-+
        +-min-+  +-,--+-----+--+--------+-+
                     +-max-+  +-,--pos-+
```

Connects all scroll bar events to one method.

Protected:

This method is protected.

Arguments:

The arguments are:

id

The ID of the scroll bar

Prog

The method that is called for all events sent by the scroll bar.

min, max

The minimum and maximum values for the scroll bar.

pos

The current or preselected value.

Return value:

-1

The specified symbolic ID could not be resolved.

0

No error.

4.2.18. AddUserMsg

```
>>-aBaseDialog~AddUserMsg(--msgToRaise--,--msgWindows----------->

>--+-------------------------------------------------------------------+-->
   +-,--filt1--+-------------------------------------------------------+-+
               +-,--wParam--+---------------------------------------+-+
                            +-,--filt2--+-------------------------+-+
                                        +-,--lParam--+----------+-+
                                                     +-,--filt3-+

>--)---------------------------------------------------------><
```

The AddUserMsg method connects a Windows message with an Object Rexx method. This method is designed to be used by ooDialog programmers who are familiar with the Windows API.

You have to pass the Windows message ID and the two message parameters (wParam and lParam) to specify the exact event you want to catch. In addition, you can specify filters for each parameter. Filters are useful for catching more than one message or one parameter with one method.

Details for all Windows messages and their parameters are available in the Windows documentation. The numeric value of the message IDs (and possibly the message parameters) can be looked up in the Windows platform SDK.

Protected:

This method is protected. You can use it only within the scope of the BaseDialog class or its subclasses.

Arguments:

The arguments are:

msgToRaise

The message that is to be sent to the Object Rexx dialog object each time the specified Windows message is caught. Provide a method with the same name. The maximum size for a message is limited to 256 characters.

msgWindows

The message in the Windows environment that is to be caught.

filt1

This filter is used to binary AND the incoming Windows message.

wParam

This is the first parameter that must be passed with the Windows message.

filt2

This filter is used to binary AND the *wParam* argument.

lParam

This is the second message parameter.

filt3

This is the filter for *lParam*.

Example:

The following example shows an implementation of the ConnectList method:

```
::class BaseDialog
      .
      .
      .
::method ConnectList
  use arg msgToRaise, id
```

```
self~AddUserMsg(msgToRaise, "0x00000111", "0xFFFFFFFF", ,
               "0x0001"||id~d2x(4), "0xFFFFFFFF", 0, 0)
```

Assume that this method is called with ID=254 and msgToRaise="ListChanged". After the ConnectList is executed, the ListChanged message is sent to the Object Rexx dialog object if the following conditions are true:

- Message "0x00000111" (WM_COMMAND) is generated by Windows in answer to an event (for example, a button is clicked or a list has changed). The filter "0xFFFFFFFF" ensures that only that message is caught; if the filter were "0xFFFFEFFF", the message "0x00001111" would be caught as well.
- The first message parameter is "0x000100FF". The first part, "0x0001", specifies the event, and the second part, "0x00FE" (equals decimal 254), specifies the dialog control where the event occurred. By using another filter it is possible to make more than one event a trigger for the ListChanged method; for example, filter "0xFFFFFFFE" would ignore the last bit of the ID, and this the same event for dialog item 255 would call ListChanged as well.
- The second message parameter and its filter are ignored.

The following example invokes a user-defined method `DoubleClick` each time the left mouse button is double-clicked:

```
self~AddUserMsg("DoubleCick","0x00000203","0xFFFFFFFF",0,0,0,0)
```

4.3. Connect Attribute Methods

The following methods create a connection between a dialog control and an attribute of the dialog object. The attribute is used to reflect the value associated with the dialog control. For instance the text of an edit control.

- For data items, such as an entry line, list box, or combo box, an attribute is created and added to the dialog object. The attribute is used as an interface to the data of the entry line, list box, or combo box.
- Check boxes and radio buttons are also data items and are therefore connected to an attribute. The only valid values for these attributes are 1 for selected and 0 for not selected.

In a UserDialog the Connect... attribute methods are called automatically from the Add... methods. The proper place for Connect... attribute methods is the InitDialog method.

4.3.1. ConnectEntryLine

```
>>-aBaseDialog~ConnectEntryLine(--id--+------------------+--)-->
                                      +-,--attributeName-+
```

The ConnectEntryLine method creates a new attribute and connects it to the entry line id. The attribute has to be synchronized with the entry line manually. This can be done globally with the SetData and GetData methods (see page GetData), or for only one item with the SetEntryLine and GetEntryLine methods (see page GetEntryLine). It is done automatically by Execute when the dialog starts and after it

terminates. If AutoDetection is enabled, or if the dialog is created dynamically (manually or based on a resource script), you do not have to use this method or any other Connect... methods that deal with dialog controls).

Arguments:

The arguments are:

id

The ID of the entry field you want to connect.

attributeName

An unused valid Rexx symbol because an attribute with exactly this name is added to the dialog object with this method. Blank spaces, ampersands (&), and colons (:) are removed from the attributeName. If this argument is omitted, is not valid, or already exists, and the ID is numeric, an attribute with the name DATAid is used, where id is the value of the first argument.

Return value:

-1

The specified symbolic ID could not be resolved.

0

No error.

Example:

In the following example, the entry line with ID 202 is associated with the attribute Name. "Put your name here!" is assigned to the newly created attribute. Then the dialog is executed. After the dialog has terminated, the data of the entry line, which the user might have changed, is copied back to the attribute Name.

```
MyDialog~ConnectEntryLine(202, "Name")
MyDialog~Name="Put your name here!"
MyDialog~Execute("SHOWTOP")
say MyDialog~Name
```

4.3.2. ConnectComboBox

```
>>-aBaseDialog~ConnectComboBox(--id--+-------------------------------------+-->
                                     +-,--+---------------+--+-----------+-+
                                          +-attributeName-+  +-,--"LIST"-+

>--)-------------------------------------------------------------><
```

The ConnectComboBox method creates an attribute and connects it to a combo box. The value of the combo box, that is, the text in the entry line or the selected list item, is associated with this attribute. See ConnectEntryLine for a more detailed description.

If the combo box is of type "Drop down list", you must specify "LIST" to connect an attribute with the combo box.

Return value:

-1

The specified symbolic ID could not be resolved.

0

No error.

4.3.3. ConnectCheckBox

```
>>-aBaseDialog~ConnectCheckBox(--id--+------------------+--)--->&lt;
                                     +-,--attributeName-+
```

The ConnectCheckBox method connects a check box control to a newly created attribute. A check box attribute has only two valid values: 1 if the box has a check mark, and 0 if it has not. See ConnectEntryLine for a more detailed description.

Return value:

-1

The specified symbolic ID could not be resolved.

0

No error.

4.3.4. ConnectRadioButton

```
>>-aBaseDialog~ConnectRadioButton(--id--+------------------+---->
                                        +-,--attributeName-+

>--)------------------------------------------------------------><
```

The ConnectRadioButton method connects a radio button control to a newly created attribute. A radio button attribute has only two valid values: 1 if the radio button is marked and 0 if it is not. See ConnectEntryLine for a more detailed description.

Return value:

-1

The specified symbolic ID could not be resolved.

0

No error.

4.3.5. ConnectListBox

```
>>-aBaseDialog~ConnectListBox(--id--+------------------+--)---->&lt;
                                    +-,--attributeName-+
```

The ConnectListBox method connects a list box to a newly created attribute. The value of the attribute is the number of the selected line. Therefore, if the attribute value is 3, the third line is currently selected or will be selected, depending on whether you set data to the dialog or receive it. See ConnectEntryLine for a more detailed description.

Return value:

-1

The specified symbolic ID could not be resolved.

0

No error.

4.3.6. ConnectMultiListBox

```
>>-aBaseDialog~ConnectMultiListBox(--id--+------------------+--->
                                         +-,--attributeName-+

>--)------------------------------------------------------------><
```

The ConnectMultiListBox method connects a list box to a newly created attribute. The list box has the multiple-selection style enabled (by setting the *MULTI* option when adding this list box), that is, you can

select more than one item at the same time. The value of the attribute is a string containing the numbers of the selected lines. The numbers are separated by blank spaces. Therefore, if the attribute value is 3 5 6, the third, fifth, and sixth item are currently selected, or will be selected if SetData is executed. See ConnectEntryLine for a more detailed description.

Return value:

-1

The specified symbolic ID could not be resolved.

0

No error.

Example:

The following example defines a list box with the name of the four seasons. It then preselects the items Summer and Winter. After execution of the dialog, it parses the value of the attribute.

```
MyDialog = .ResDialog~new(...)
MyDialog~NoAutoDetection
MyDialog~AddListBox(205, ..., "MULTI")
MyDialog~ConnectMultiListBox(205, "ListBox")
seasons.1="Spring"
seasons.2="Summer"
seasons.3="Autumn"
seasons.4="Winter"
do season over seasons
  MyDialog~AddListEntry(205, season)
end
MyDialog~ListBox="2 4"
MyDialog~Execute("SHOWTOP")
selItems = MyDialog~ListBox
do until anItem =""
   parse var selItems anItem selItems
   say "You selected: "seasons.anItem
end
```

4.3.7. AddAttribute

```
>>-aBaseDialog~AddAttribute(--id--+-------------------+--)------><
                                  +-,--attributeName-+
```

The AddAttribute method adds an attribute to the dialog object. The attribute is associated with the dialog control *id*.

Protected:

This method is for internal use only.

Arguments:

The arguments are:

id

The ID of the dialog control.

attributeName

The name you want to give to the attribute. This name must comply with the conventions of Object Rexx for valid symbols. AddAttribute checks whether the argument is valid. In case of an invalid argument, an attribute with the name DATA*id* is created, where *id* is the value of the first argument. This method automatically removes blanks, ampersands (&), and colons (:).

Example:

The first and second lines generate the attributes `Add` and `List all items`. The third line generates the assembled attribute `DATA34` because `ListALLitems` already exists. The fourth line creates attribute `DATA35` because `Update+Refresh` is not a valid symbol name.

```
self~AddAttribute(32, "&Add")
self~AddAttribute(33, "List all items")
self~AddAttribute(34, "ListALLitems:")
self~AddAttribute(35, "Update+Refresh")
```

4.4. Get and Set Methods

Get methods are used to retrieve the data from all or individual controls of a dialog. Set methods are used to set the values of all or individual controls, without changing the associated Object Rexx attributes.

4.4.1. GetData

```
>>-aBaseDialog~GetData----------------------------------------><
```

The GetData method receives data from the Windows dialog and copies it to the associated object attributes.

Example:

The following example continues the SetData example:

```
      .
      .
      .
MyDialog~ConnectEntryLine(102, "ENTRYLINE_1")
```

```
MyDialog~ConnectCheckBox(201, )
MyDialog~ConnectListbox(203, "LISTBOX_DAYS")
     .
     .
     .
/* process the dialog */
     .
     .
     .
MyDialog~GetData            /* retrieve dialog item value */
say MyDialog~ENTRYLINE_1
say MyDialog~DATA201
say MyDialog~LISTBOX_DAYS
```

4.4.2. SetData

```
>>-aBaseDialog~SetData----------------------------------------><
```

The SetData method transfers data from the Object Rexx attributes to the Windows dialog.

Example:

Dialog items with ID 102, 201 and 203 are connected to the attributes `ENTRYLINE_1`, `DATA201`, and `LISTBOX_DAYS`. Attribute `DATA201` is generated by the ConnectCheckBox method. Then the attributes are initialized with some values. This does not change the dialog window, unless you run the SetData method.

```
     .
     .
     .
MyDialog~ConnectEntryLine(102, "ENTRYLINE_1")
MyDialog~ConnectCheckBox(201,)
MyDialog~ConnectListbox(203, "LISTBOX_DAYS")
     .
     .
     .
MyDialog~ENTRYLINE_1="Memorial Day"
MyDialog~DATA201=1
MyDialog~LISTBOX_DAYS="Monday"

MyDialog~SetData
```

4.4.3. ItemTitle

```
>>-aBaseDialog~ItemTitle(--id--)------------------------------><
```

The ItemTitle method returns the title of the given dialog item.

Arguments:

The only argument is:

id

The ID of the dialog item.

4.4.4. SetStaticText

```
>>-aBaseDialog~SetStaticText(--id--,--aString--)--------------><
```

The SetStaticText method changes the text of a static text control.

Arguments:

The arguments are:

id

The ID of the static text control for which you want to change the text.

aString

The new text for the static text control.

4.4.5. GetEntryLine

```
>>-aBaseDialog~GetEntryLine(--id--)---------------------------><
```

The GetEntryLine method returns the value of the given entry line.

Arguments:

The only argument is:

id

The ID of the entry line.

4.4.6. SetEntryLine

```
>>-aBaseDialog~SetEntryLine(--id--,--aString--)----------------><
```

The SetEntryLine method puts the value of a string into an entry line.

Arguments:

The arguments are:

id

The ID of the entry line.

aString

The value to be assigned to the entry line.

Example:

Assume that three methods are connected to push buttons. The SetToDefault method overrides the value in the Windows dialog entry line 234 with the value 256 but does not change its associated attribute. Using SetEntryLine has the same effect as a change to the entry line made by the user. The associated attribute in the Object Rexx object (`DATA234`) still has the original value. Thus it is possible to undo the changes or confirm them.

```
::method SetToDefault
   self~SetEntryLine(234, "256")

::method AcceptValues
   self~GetAttrib(DATA234)

::method UndoChanges
   self~SetAttrib(DATA234)
```

4.4.7. GetListLine

```
>>-aBaseDialog~GetListLine(--id--)----------------------------><
```

The GetListLine method returns the value of the currently selected list item. If you need the index of the item, use the GetCurrentListIndex method. If no item is selected, a null string is returned.

Arguments:

The only argument is:

id

The ID of the list box.

4.4.8. SetListLine

```
>>-aBaseDialog~SetListLine(--id--,--aString--)-----------------><
```

The SetListLine method assigns the value of a string to the list box. Thus the item with the value of *aString* becomes selected. The first item is selected if the string is not found in the list box. This method does not apply to a multiple selection list box (see SetMultiList).

Arguments:

The arguments are:

id

The ID of the list box.

aString

The value of the item to be selected.

Example:

The following example selects item "New York" in list box 232:

```
MyBaseDialog~SetListLine(232, "New York")
```

4.4.9. GetMultiList

```
>>-aBaseDialog~GetMultiList(--id--)---------------------------><
```

The GetMultiList method can be applied to a multiple-selection list box. It returns a string containing the indexes of up to 1000 selected items. The numbers are separated by blanks.

Arguments:

The only argument is:

id

The ID of the multiple-selection list box.

Example:

The following example shows how to handle a multiple-selection list box. It parses the returned string as long as it contains an index.

```
selLines = MyDialog~GetMultiList(555)
do until selLines = ""
   parse var selLines aLine selLines
   say aLine
end
```

4.4.10. SetMultiList

```
>>-aBaseDialog~SetMultiList(--id--,--data--)------------------><
```

The SetMultiList method selects one or more lines in a multiple-selection list box.

Arguments:

The arguments are:

id

The ID of the multiple-selection list box.

data

The indexes (separated by blanks) of the lines to be selected.

Example:

The following example selects the lines 2, 5, and 6 of list box 345:

```
MyDialog~SetMultiList(345, "2 5 6")
```

4.4.11. GetComboLine

```
>>-aBaseDialog~GetComboLine(--id--)---------------------------><
```

The GetComboLine method returns the value of the currently selected list item of a combo box. If you need the index of the item, use the GetCurrentComboIndex method. If no item is selected, a null string is returned.

Arguments:

The only argument is:

id

The ID of the combo box.

4.4.12. SetComboLine

```
>>-aBaseDialog~SetComboLine(--id--,--aString--)----------------><
```

The SetComboLine method assigns a string to the given combo box. Thus the item with the value of *aString* becomes selected. If *aString* is not found in the combo box, then the first item is selected.

Arguments:

The arguments are:

id

The ID of the combo box.

aString

The value of the item to be selected.

4.4.13. GetRadioButton

```
>>-aBaseDialog~GetRadioButton(--id--)-------------------------><
```

The GetRadioButton method returns 1 if the radio button is selected, 0 if it is not selected.

Arguments:

The only argument is:

id

The ID of the radio button.

4.4.14. SetRadioButton

```
>>-aBaseDialog~SetRadioButton(--id--,--data--)-----------------><
```

The SetRadioButton method marks the radio button if the given data value is 1, and removes the mark if the value is 0.

Arguments:

The arguments are:

id

The ID of the radio button.

data

1 to select the button or 0 to deselect it.

4.4.15. GetCheckBox

```
>>-aBaseDialog~GetCheckBox(--id--)----------------------------><
```

The GetCheckBox method returns the value of a check box: 1 if the check box is selected (has a check mark), 0 if it is not selected.

Arguments:

The only argument is:

id

The ID of the check box.

4.4.16. SetCheckBox

```
>>-aBaseDialog~SetCheckBox(--id--,--data--)--------------------><
```

The SetCheckBox method puts a check mark in the check box if the given data value is 1 and removes the check mark if the value is 0.

Arguments:

The arguments are:

id

The ID of the check box.

data

The value 1 to check the box or 0 to remove the check mark.

4.4.17. GetValue

```
>>-aBaseDialog~GetValue(--id--)--------------------------------><
```

The GetValue method gets the value of a dialog item, regardless of its type. The item must have been connected before.

Arguments:

The only argument is:

id

The ID of the dialog item.

4.4.18. SetValue

```
>>-aBaseDialog~SetValue(--id--,--dataString--)----------------><
```

The SetValue method sets the value of a dialog item. You do not have to know what kind of item it is. The dialog item must have been connected before.

Arguments:

The arguments are:

id

The ID of the dialog item.

dataString

The value that is assigned to the item. It must be a valid value.

Example:

The following example sets dialog item 123 to (string) value "1 2 3". This is meaningful if 123 is an entry field, or if it is a list box that contains the line "1 2 3". However, it is an error to apply this against a check box.

```
MyDialog~SetValue(123, "1 2 3")
```

Note: If it is a multiple-selection list box, the SetValue method does not look for an item with "1 2 3" as value but highlights the first, second, and third line.

4.4.19. GetAttrib

```
>>-aBaseDialog~GetAttrib(--attributeName--)-------------------><
```

The GetAttrib method assigns the value of a dialog item to the associated Object Rexx attribute. It does not return a value. You do not have to know the ID or the type of the dialog item.

Arguments:

The only argument is:

attributeName

The name of the attribute.

Example:

The following example shows how to get the data value of a dialog item without knowing its ID:

```
MyDialog~GetAttrib("FirstName")
if MyDialog~FirstName\="" then ...
```

4.4.20. SetAttrib

```
>>-aBaseDialog~SetAttrib(--attributeName--)-------------------><
```

The SetAttrib method copies the value of an attribute to the associated dialog item. You do not have to know the ID or the type of the dialog item.

Arguments:

The only argument is:

attributeName

The name of the attribute.

Example:

The following example copies the value of the attribute DATA101 to the associated dialog item:

```
MyBaseDialog~SetAttrib("DATA101")
```

4.4.21. SetDataStem

```
>>-aBaseDialog~SetDataStem(--dataStem.--)----------------------><
```

The SetDataStem method sets all Windows dialog items to the values within the given stem; the suffixes of the stem variable are the dialog IDs.

Protected:

This method is protected.

Arguments:

The only argument is:

dataStem.

A stem variable containing initialization data. Remember the trailing period.

Example:

The following example initializes the dialog items with ID 21, 22, and 23:

```
      .
      .
      .
dlgStem.21="Windows 95"
dlgStem.22="0"
dlgStem.23="1 2 3"
self~SetDataStem(dlgStem.)
```

4.4.22. GetDataStem

```
>>-aBaseDialog~GetDataStem(--dataStem.--)---------------------><
```

The GetDataStem method gets the values of all dialog items and copies them to the given stem.

Protected:

This method is protected.

Arguments:

The only argument is:

dataStem.

The name of a stem variable to which the data is returned. Remember the trailing period.

4.5. Standard Event Methods

The following methods are abstract methods that are called each time a push button with ID 1, 2, or 9 is pressed.

4.5.1. OK

```
>>-aBaseDialog~OK---------------------------------------------><
```

The OK method is called in response to a pressed OK button. It calls Validate to get its return code. The default return code is the self~finished attribute, which is usually 1, in which case the dialog is terminated.

Protected:

This method is protected. You might want to override it in your subclass. If you do, forward the OK message to the parent class after processing has finished. Set the self~finished attribute to 1 or 0 and return it. The dialog continues executing if you return the value 0. See also Validate.

Example:

The following example shows how to override the OK method:

```
::method OK
...
self~ok:super()
self~finished = 1
return self~finished
```

4.5.2. Cancel

```
>>-aBaseDialog~Cancel------------------------------------------><
```

The Cancel method is called in response to a pressed Cancel button. The default return code is the self~finished attribute, which is usually 1, in which case the dialog is terminated. The InitCode attribute is set to 2 if the dialog is terminated.

Protected:

This method is protected. You might want to override it in your subclass. If you do, forward the Cancel message to the parent class after processing has finished. Set the self~finished attribute to 1 or 0 and return it. The dialog continues executing if you return the value 0.

4.5.3. Help

```
>>-aBaseDialog~Help--------------------------------------------><
```

The Help method is called in response to a pressed Help button.

Protected:

This method is protected. You might want to override it in your subclass.

4.5.4. Validate

```
>>-aBaseDialog~Validate---------------------------------------><
```

The Validate method is an abstract method that is called to determine whether the dialog can be closed. This method is called by the OK method. The standard implementation is that Validate returns 1 and the dialog is closed. The dialog is not closed if Validate returns 0.

Protected:

The method is designed to be defined in a subclass.

Example:

In the following example Validate checks whether entry line 203 is empty. If it is empty, Validate returns 0, which indicates that the dialog cannot be closed.

```
::class MyDialog subclass UserDialog
::method Validate
   if self~GetEntryLine(203) = "" then return 0
   else return 1
```

4.5.5. Leaving

```
>>-aBaseDialog~Leaving----------------------------------------><
```

The Leaving method is called when the dialog was closed.

4.5.6. DeInstall

```
>>-aBaseDialog~DeInstall--------------------------------------><
```

The DeInstall method removes the external functions from the Object Rexx API manager. It should be called at the end of each dialog. The installed functions are freed when all dialogs are finished.

4.6. Combo Box Methods

The following methods belong to combo boxes.

4.6.1. AddComboEntry

```
>>-aBaseDialog~AddComboEntry(--id--,--aString--)---------------><
```

The AddComboEntry method adds a string to the list of a combo box. The new item becomes the last one, if the list does not have the SORT flag set. In the case of a sorted list, the new item is inserted at the proper position.

Arguments:

The arguments are:

id

The ID of a combo box.

aString

The data to be inserted as a new line.

Example:

The following example adds the new line, `Another item`, to the list of combo box 103:

```
MyDialog~AddComboEntry(103, "Another item")
```

4.6.2. InsertComboEntry

```
>>-aBaseDialog~InsertComboEntry(--id--,--+-------+--,--string--)-><
                                         +-index-+
```

The InsertComboEntry method inserts a string into the list of a combo box.

Arguments:

The arguments are:

id

The ID of the combo box.

index

The index (line number) where you want to insert the new item. If this argument is omitted, the new item is inserted after the currently selected item.

string

The data string to be inserted.

Example:

This statement inserts `The new third line` after the second line into the list of combo box 103:

```
MyDialog~InsertComboEntry(103, 2, "The new third line")
```

4.6.3. DeleteComboEntry

```
>>-aBaseDialog~DeleteComboEntry(--id--,--index--)-------------><
```

The DeleteComboEntry method deletes a string from the combo box.

Arguments:

The arguments are:

id

The ID of the combo box.

index

The line number of the item to be deleted. Use the FindComboEntry method (see pageFindComboEntry) to retrieve the index of an item.

Example:

The following example shows a method that deletes the item that is passed to the method in the form of a text string from combo box 203:

```
      .
      .
      .
::method DeleteFromCombo
   use arg delStr
   idx = self~FindComboEntry(203, delStr)
   self~DeleteComboEntry(203, idx)
```

4.6.4. FindComboEntry

```
>>-aBaseDialog~FindComboEntry(--id--,--aString--)-------------><
```

The FindComboEntry method returns the index corresponding to a given text string in the combo box.

Arguments:

The arguments are:

id

The ID of the combo box

aString

The string of which you search the index in the combo box.

Example:

See DeleteComboEntry for an example.

4.6.5. GetComboEntry

```
>>-aBaseDialog~GetComboEntry(--id--,--index--)----------------><
```

The GetComboEntry method returns the string at index of the combo box.

Arguments:

The arguments are:

id

The ID of the combo box

index

The index of the list entry to be retrieved

Example:

```
if dlg~GetComboEntry(203,5)="JOHN"
then ...
```

4.6.6. GetComboItems

```
>>-aBaseDialog~GetComboItems(--id--)--------------------------><
```

The GetComboItems method returns the number of items in the combo box.

Arguments:

The only argument is:

id

The ID of the combo box

4.6.7. GetCurrentComboIndex

```
>>-aBaseDialog~GetCurrentComboIndex(--id--)--------------------><
```

The GetCurrentComboIndex method returns the index of the currently selected item within the list. See GetComboLine for information on how to retrieve the selected combo box item.

Arguments:

The only argument is:

id

The ID of the combo box.

Example:

The following example displays the line number of the currently selected combo box item within entry line 240:

```
::class MyListDialog subclass UserDialog
      .
      .
      .
::method Init
   self~Init:super
   self~ConnectList(230, "ListSelected")
      .
      .
      .
::method ListSelected
   line = self~GetCurrentComboIndex(230)
   SetEntryLine(240, line)
```

Method `ListSelected` is called each time the selected item within the combo box changes.

4.6.8. SetCurrentComboIndex

```
>>-aBaseDialog~SetCurrentComboIndex(--id--+----------+--)------><
                                          +-,--index-+
```

The SetCurrentComboIndex method selects the item with the given index within the list. If called without an index, all items in the list are deselected. See SetComboLine for information on how to select a combo box item using a data value.

Arguments:

The arguments are:

id

The ID of the combo box.

index

The index within the combo box.

4.6.9. ChangeComboEntry

```
>>-aBaseDialog~ChangeComboEntry(--id--,--+-------+--,--aString--)-><
                                         +-index-+
```

The ChangeComboEntry method changes the value of a given entry in a combo box to a new string.

Arguments:

The arguments are:

id

The ID of the combo box

index

The index number of the item you want to replace. To retrieve the index, use the FindComboEntry or GetCurrentComboIndex method (see page FindComboEntry or GetCurrentComboIndex).

aString

The new text.

Example:

In the following example, method `ChangeButtonPressed` changes the currently selected line of combo box 230 to the value in entry line 250:

```
      .
      .
      .
::method ChangeButtonPressed
   idx = self~GetCurrentComboEntry(230)
   str = self~GetEntryLine(250)
   self~ChangeComboEntry(230, idx, str)
```

4.6.10. ComboAddDirectory

```
>>-aBaseDialog~ComboAddDirectory(--id--,--drvpath--,------------->

      +---------------+
      V               |
>--"----+-READWRITE-+-+--"--)----------------------------------><
        +-READONLY--+
```

```
+-HIDDEN----+
+-SYSTEM----+
+-DIRECTORY-+
+-ARCHIVE---+
```

The ComboAddDirectory method adds all or selected file names in the given directory to the combo box.

Arguments:

The arguments are:

id

The ID of the combo box.

drvpath

The drive, path, and name pattern.

fileAttributes

Specify the file attributes that the files must have in order to be added:

READWRITE

Normal read/write files (same as none).

READONLY

Files that have the read-only bit.

HIDDEN

Files that have the hidden bit.

SYSTEM

Files that have the system bit.

DIRECTORY

Files that have the directory bit.

ARCHIVE

Files that have the archive bit.

Example:

The following example fills the combo box list with the names of all read/write files with extension .REX in the given directory:

```
MyDialog~ComboAddDirectory(203, drive":\"path"\*.rex", "READWRITE")
```

4.6.11. ComboDrop

```
>>-aBaseDialog~ComboDrop(--id--)------------------------------><
```

The ComboDrop method deletes all items from the list of the given combo box.

Arguments:

The only argument is:

id

The ID of the combo box.

4.7. List Box Methods

The following methods deal with list boxes.

4.7.1. GetListWidth

```
>>-aBaseDialog~GetListWidth(--id--)----------------------------><
```

The GetListWidth method returns the scrollable width of a list box, in dialog units.

Arguments:

The only argument is:

id

The ID of the list box of which you want to know the scrollable width.

Return value:

The width of the scrollable area of the list box, in dialog units.

4.7.2. SetListWidth

```
>>-aBaseDialog~SetListWidth(--id--,--scrollwidth--)------------><
```

The SetListWidth method sets the scrollable width of a list box, in dialog units. If the scrollable width is greater than the width of the list box and the "HSCROLL" (WS_HSCROLL in the resource script) style is defined for the list box (see AddListBox), a horizontal scroll bar is displayed.

Arguments:

The arguments are:

id

The ID of the list box for which you want to set the scrollable width.

scrollwidth

The width of the scrollable area of the list box, in dialog units.

Return value:

This method does not return a value.

4.7.3. SetListColumnWidth

```
>>-aBaseDialog~SetListColumnWidth(--id--,--columnwidth--)------><
```

The SetListColumnWidth method sets the width of all columns in a list box, in dialog units.

Arguments:

The arguments are:

id

The ID of the list box for which you want to set the column width.

columnwidth

The width of the columns in the list box, in dialog units.

Return value:

This method does not return a value.

4.7.4. AddListEntry

```
>>-aBaseDialog~AddListEntry(--id--,--aString--)---------------><
```

The AddListEntry method adds a string to the given list box. See also AddComboEntry. The line is added at the end (by default), or in sorted order if the list box was defined with the sorted flag.

Arguments:

The arguments are:

id

The ID of a list box.

aString

The data to be inserted as a new line.

4.7.5. InsertListEntry

```
>>-aBaseDialog~InsertListEntry(--id--,--+-------+--,--aString--)-><
                                        +-index-+
```

The InsertListEntry method inserts a string into the given list box. See also InsertComboEntry.

Arguments:

The arguments are:

id

The ID of the list box.

index

The index (line number starting with 1) of the item after which the new item is inserted. If this argument is omitted, the new item is inserted after the currently selected item.

aString

The text string to be inserted.

4.7.6. DeleteListEntry

```
>>-aBaseDialog~DeleteListEntry(--id--,--index--)--------------><
```

The DeleteListEntry method deletes an item from a list box. See also DeleteComboEntry.

Arguments:

The arguments are:

id

The ID of the list box.

index

The line number of the item to be deleted. Use FindListEntry to retrieve the index of an item. If this argument is omitted, the currently selected item is deleted.

4.7.7. FindListEntry

```
>>-aBaseDialog~FindListEntry(--id--,--aString--)--------------><
```

The FindListEntry method returns the index of the given string within the given list box. The first item has index 1, the second has index 2, and so forth. If the list box does not contain the string, 0 is returned.

Arguments:

The arguments are:

id

The ID of the list box.

aString

The item text you are looking for.

Example:

The following example shows a method that adds the contents of an entry line (214) to the list box (215) if no item with the same value is already contained in it:

```
      .
      .
      .
::method PutEntryInList
   str = self~GetEntryLine(214)
   if self~FindListEntry(215, str) = 0 then
      self~AddListEntry(215, str)
```

4.7.8. GetListEntry

```
>>-aBaseDialog~GetListEntry(--id--,--index--)-----------------><
```

The GetListEntry method returns the string at index of the list.

Arguments:

The arguments are:

id

The ID of the list box.

index

The index of the list entry to be retrieved.

Example:

```
if dlg~GetListEntry(203,5)="JOHN"
then ...
```

4.7.9. GetListItems

```
>>-aBaseDialog~GetListItems(--id--)---------------------------><
```

The GetListItems method returns the number of items in the list box.

Arguments:

The only argument is:

id

The ID of the list box.

4.7.10. GetListItemHeight

```
>>-aBaseDialog~GetListItemHeight(--id--)----------------------><
```

The GetListItemHeight method returns the height of the items in a list box, in dialog units.

Arguments:

The only argument is:

id

The ID of the list box of which you want to know the item height.

Return value:

The height of the list box items, in dialog units.

4.7.11. SetListItemHeight

```
>>-aBaseDialog~SetListItemHeight(--id--,--itemheight--)-------><
```

The SetListItemHeight method sets the height for all items in a list box, in dialog units. It determines the space between the individual list box items.

Arguments:

The arguments are:

id

The ID of the list box for which you want to set the item height.

itemheight

The height of the items in the list box, in dialog units.

Return value:

A number smaller than 0 if the height that you specify is not valid.

4.7.12. GetCurrentListIndex

```
>>-aBaseDialog~GetCurrentListIndex(--id--)--------------------><
```

The GetCurrentListIndex method returns the index of the currently selected list box item, or 0 if no item is selected. See GetListLine for information on how to retrieve the selected list box item.

Arguments:

The only argument is:

id

The ID of the list box.

4.7.13. SetCurrentListIndex

```
>>-aBaseDialog~SetCurrentListIndex(--id--+----------+--)------><
                                         +-,--index-+
```

The SetCurrentListIndex selects the item with the given index in the list. If called without an index, all items in the list are deselected. See SetListLine for information on how to select a list box item using a data value.

Arguments:

The arguments are:

id

The ID of the list box.

index

The index within the list box.

4.7.14. ChangeListEntry

```
>>-aBaseDialog~ChangeListEntry(--id--,--+-------+--,--aString--)-><
                                        +-index-+
```

The ChangeListEntry method changes the contents of a line in a list box.

Arguments:

The arguments are:

id

The ID of the list box.

index

The index of the item that you want to replace. If this argument is omitted, the currently selected item is changed.

aString

The new text of the item.

4.7.15. SetListTabulators

```
                                        +-,---+
                                        V     |
>>-aBaseDialog~SetListTabulators(--id--,----tab-+--)----------><
```

The SetListTabulators method sets the tabulators for a list box. Thus you can use items containing tab characters ("09"x), which is useful for formatting the list in more than one column.

Arguments:

The arguments are:

id

The ID of the list box.

tab

The positions of the tabs relative to the left edge of the list box.

Example:

The following example creates a four-column list and adds a tab-formatted row to the list. The tabulator positions are 10, 20, and 30.

```
MyDialog~SetListTabulators(102, 10, 20, 30)
MyDialog~AddListEntry(102, var1  "09"x
var2  "09"x  ,
                          var3  "09"x  var4)
```

4.7.16. ListAddDirectory

```
                                                        +---------------+
                                                        V               |
>>-aBaseDialog~ListAddDirectory(--id--,--drvPath--,--"----+-READWRITE-+-+--"--)-><
                                                          +-READONLY--+
                                                          +-HIDDEN----+
                                                          +-SYSTEM----+
                                                          +-DIRECTORY-+
                                                          +-ARCHIVE---+
```

The ListAddDirectory method adds all or selected file names of a given directory to the list box. See ComboAddDirectory for more information.

4.7.17. ListDrop

```
>>-aBaseDialog~ListDrop(--id--)------------------------------><
```

The ListDrop method removes all items from the list box.

Arguments:

The only argument is:

id

The ID of the list box.

4.8. Scroll Bar Methods

The following methods are used to set or get the behavior of a scroll bar. You can connect scroll bars with numerical entry fields to edit the value with the mouse.

4.8.1. GetSBRange

```
>>-aBaseDialog~GetSBRange(--id--)----------------------------><
```

The GetSBRange method returns the range of a scroll bar control. It returns the two values (minimum and maximum) in one string, separated by a blank.

Protected:

This method is protected.

Arguments:

The only argument is:

id

The ID of the scroll bar.

Example:

The following example demonstrates how to get the minimum and the maximum values of the scroll bar:

```
      .
      .
      .
::method DumpSBRange
SBrange = self~GetSBRange(234)
parse var SBrange SBmin SBmax
say SBmin " - " SBmax
```

4.8.2. SetSBRange

```
                                                +-1------+
>>-aBaseDialog~SetSBRange(--id--,--min--,--max--,--+--------+--)-><
                                                +-redraw-+
```

The SetSBRange method sets the range of a scroll bar control. It sets the minimum and maximum values.

Protected:

This method is not intended to be used outside of the BaseDialog class.

Arguments:

The arguments are:

id

The ID of a scroll bar control.

min

The minimum value.

max

The maximum value.

redraw

A flag indicating whether (1) or not (0) the scroll bar should be redrawn. The default is 1.

Example:

The following example allows the scroll bar to take values between 1 and 10:

```
MyDialog~SetSBRange(234, 1, 10, 1)
```

4.8.3. GetSBPos

```
>>-aBaseDialog~GetSBPos(--id--)------------------------------><
```

The GetSBPos method returns the current value of a scroll bar control.

Arguments:

The only argument is:

id

The ID of the scroll bar.

4.8.4. SetSBPos

```
                                        +-1------+
>>-aBaseDialog~SetSBPos(--id--,--pos--,--+--------+--)--------><
                                        +-redraw-+
```

The SetSBPos method sets the current value of a scroll bar control.

Protected:

This method is protected.

Arguments:

The arguments are:

id

The ID of the scroll bar.

pos

The value to which you want to set the scroll bar. It must be within the defined range.

redraw

A flag indicating whether (1) or not (0) the scroll bar should be redrawn. The default is 1.

4.8.5. CombineELwithSB

```
>>-aBaseDialog~CombineELwithSB(--elid--,--sbid--+-------------------------+--)-><
                                                +-,--+------+--+--------+-+
                                                     +-step-+  +-,--pos-+
```

The CombineELwithSB method connects an entry line with a scroll bar such that each time the slider of the scroll bar is moved, the value of the entry field is changed. This method must be used in a method registered with ConnectScrollBar.

Arguments:

The arguments are:

elid

The ID of the entry line.

sbid

The ID of the scroll bar.

step

The size of one step. If, for example, step is 3 and the current position is 4, the next position is 7.

pos

If the step value is zero, this sets the position of the scroll bar and entry line. Use it in the method registered for drag.

Example:

The following example continues the example of ConnectScrollBar. In the registered methods an entry line (251) is combined with the scroll bar (255).

```
::method Increase
self~CombineELwithSB(251,255,+20)
::method Decrease
self~CombineELwithSB(251,255,-20)
::method Drag
use arg wparam, lparam                  /* wparam=position */
self~CombineELwithSB(251,255,0,wparam)
```

4.8.6. DetermineSBPosition

```
>>-aBaseDialog~DetermineSBPosition(--id--,--posdata--+----------------------------+--)-><
                                                     +-,--+--------+--+---------+-+
                                                          +-single-+  +-,--page-+
```

The DetermineSBPosition method calculates and sets the new scroll bar position based on the position data retrieved from the scroll bar and the step information.

Protected:

This method is protected.

Arguments:

The arguments are:

id

The ID of the scroll bar.

posdata

The position information sent with the connected scroll bar event.

single

This number is added (or subtracted if negative) to the current position for a single step. If omitted, the single step size is 1.

page

This number is added (or subtracted if negative) to the current position for a page step. If omitted, the page step size is 10.

Return value:

The new scroll bar position.

Example:

The following example demonstrates how to update the scroll bar position. Each time the ScrollBarEventHandler is called by an event for scroll bar SB_SIZE, the position of the scroll bar is calculated and updated. `posdata` is sent along with the scroll bar event.

```
   /* Method ScrollBarEventHandler is connected to item SB_SIZE */
::method ScrollBarEventHandler
    use arg posdata, sbwnd
    pos = self~DetermineSBPosition("SB_SIZE",posdata,1,25)
    return pos
```

4.9. Methods to Query Operating System Values

The methods in this section return information known to, or used by, the operating system when it works with dialogs or dialog controls. This includes things like window handles, window sizes, and window positions. It also includes things like the width of a scroll bar or the number of monitors attached to the system.

4.9.1. GetSelf

```
>>-aBaseDialog~GetSelf---------------------------------------><
```

The GetSelf method returns the handle of the Windows Dialog associated with the ooDialog dialog instance. A handle is a unique reference to a particular Windows object. Handles are used within some of the methods to work on a specific Windows object.

> **Note:** Prior to ooRexx 3.2.0, and all the way back through the original IBM documentation, the `GetSelf` method was not documented. And, the Get method was documented as through it was the `GetSelf` method. This most likely has caused confusion. The `Get` method returns the handle to the most recently created dialog. In an application with more than one dialog this may not be the same as the Windows dialog associated with the instance object the programmer is working with.

Example:

Below is an example demonstrating these Windows handles and how they are related. Assume `showHandles` is a method of a subclass of BaseDialog and that staticDlg is a saved reference to another ooDialog object and that the static dialog was just created. The `showHandles` method is connected to a push button and then displays the handles when the user clicks that button.

```
::method showHandles
  expose staticDlg

  hwndTop = self~get
  hwndMyDialogHandle = self~dlgHandle
```

```
hwndMySelf = self~getSelf
hwndStatic = staticDlg~dlgHandle
say "Top dialog:      " hwndTop
say "Self (dlgHandle):" hwndMyDialogHandle
say "Self (getSelf):  " hwndMySelf
say "Static dialog:   " hwndStatic
```

Output:

The above might write to the console something like the following:

```
Top dialog:       787096
Self (dlgHandle): 4522228
Self (getSelf):   4522228
Static dialog:    787096
```

4.9.2. Get

```
>>-aBaseDialog~Get-------------------------------------------><
```

The `Get` method returns the handle of the Windows Dialog associated with the top ooDialog dialog. A handle is a unique reference to a particular Windows object. Handles are used within some of the methods to work on a specific Windows object.

Note: If more than one ooDialog exists, the top dialog and the dialog whose method is executing may not be the same. When more than one dialog exists, the top dialog is the one that was created last.

Example:

Below is an example demonstrating that the top dialog and the executing dialog may not be the same. Assume display is a method of a subclass of BaseDialog and that staticDlg is a saved reference to another ooDialog object.

```
::method display
  expose staticDlg

  hwndTop = self~get
  hwndSelf = self~dlgHandle
  hwndStatic = staticDlg~dlgHandle
  say "Top dialog:   " hwndTop
  say "Self:         " hwndSelf
  say "Static dialog:" hwndStatic
```

Output:

The above might write to the console something like the following:

```
Top dialog:    787096
Self:          4522228
Static dialog: 787096
```

4.9.3. GetItem

```
>>-aBaseDialog~GetItem(--id--+---------+--)------------------><
                             +-,--hDlg-+
```

The GetItem method returns the handle of a particular dialog item.

Arguments:

The arguments are:

id

The ID of the dialog element.

hDlg

The handle of the dialog. If it is omitted, the main dialog handle is used.

Example:

The following example returns the handle of a push button:

```
hndPushButton = MyDialog~GetItem(101)
```

4.9.4. GetControlID

```
>>-aBaseDialog~GetControlID(--hWnd--)------------------------><
```

Given a valid window handle to a dialog control, the `GetControlID` method returns the resource ID of the control.

Arguments:

The only argument is:

hWnd

The window handle of the dialog control.

Return value:

Negative values indicate an error.

-1

The `hWnd` argument is not a valid window handle.

less than -1

The value is the negated Operating System Error code. The absolute value of the return can be used to look up the error reason in the Windows documentation.

other

The resource ID of the dialog control.

Example:

The following is a complete working example. It allows the user to check a check box with the F2 key and uncheck it with the F3 key. The F2 and F3 key press events are connected to the `check` and `uncheck` methods. When the user presses the F2 key, the program determines which control has the focus, uses the returned window handle to get the control ID, and then uses the ID to check the check box. And the same approach if the F3 key is pressed.

Note that in this dialog, there are only four controls that can have the focus. If the OK button has the focus, then nothing is done. In a more complex application, the programmer would probably check that the resource ID matches one of the check boxes instead.

```
/* Simple Grocery List */

  dlg = .SimpleDialog~new
  dlg~constDir[IDC_GB_LIST] = 101
  dlg~constDir[IDC_CB_MILK] = 102
  dlg~constDir[IDC_CB_BREAD] = 103
  dlg~constDir[IDC_CB_FRUIT] = 104
  dlg~constDir[IDC_CB_CEREAL] = 105

  if dlg~initCode = 0 then do
    dlg~create(30, 30, 150, 150, "The Simple Grocery List", "VISIBLE")
    dlg~Execute("SHOWTOP")
    dlg~Deinstall
  end

-- End of entry point.
::requires "oodWin32.cls"

::class SimpleDialog subclass UserDialog inherit AdvancedControls MessageExtensions

::method check

  hWnd = self~getFocus
  id = self~getControlID(hWnd)
  if id == self~constDir[IDOK] then return

  self~getCheckControl(id)~check

::method unCheck

  id = self~getControlID(self~getFocus)
```

```
    if id == self~constDir[IDOK] then return

    self~getCheckControl(id)~uncheck

  ::method defineDialog

    self~addGroupBox(10, 20, 130, 90, "Check Needed Groceries", "", IDC_GB_LIST);
    self~addCheckBox(IDC_CB_MILK, , 30, 35, , , "Milk", "GROUP");
    self~addCheckBox(IDC_CB_BREAD, , 30, 55, , , "Bread", "NOTAB");
    self~addCheckBox(IDC_CB_FRUIT, , 30, 75, , , "Fruit", "NOTAB");
    self~addCheckBox(IDC_CB_CEREAL, , 30, 95, , , "Cereal", "NOTAB");

    self~addButton(IDOK, 105, 120, 35, 15, "OK", , "GROUP")

  ::method initDialog

    -- We know that the key code for F2 is 113 so we don't need to use the
    -- VirtualKeyCodes class.  We use the 'none' filter so that only a F2 or F3
    -- key press is captured.  (Not Alt-F2, or Shift-F2, etc..)
    self~connectKeyPress(check, 113, "NONE")
    self~connectKeyPress(unCheck, 114, "NONE")
```

4.9.5. GetPos

```
>>-aBaseDialog~GetPos----------------------------------------><
```

The GetPos method returns the dialog window's position in pixels. The values are returned in a string separated by blanks for parsing.

Example:

The following example moves the window towards the top left of the screen.

```
parse value self~GetPos with px py
self~Move( px%self~FactorX - 10, py%self~FactorY - 10)
```

4.9.6. GetButtonRect

```
>>-aBaseDialog~GetButtonRect(--id--)--------------------------><
```

The GetButtonRect method returns the size and position of the given button. The four values (left, top, right, bottom) are returned in one string separated by blanks.

Arguments:

The only argument is:

id

The ID of the button

4.9.7. GetWindowRect

```
>>-aBaseDialog~GetWindowRect(--hwnd--)------------------------><
```

The GetWindowRect method returns the size and position of the given window. The four values (left, top, right, bottom) are returned in one string separated by blanks.

Arguments:

The only argument is:

hwnd

The handle of the window. There are a number of ways to get the window handle of objects used in ooDialog. For instance, use the GetSelf method to retrieve the window handle of the dialog instance. Use the Get method to retrieve the window handle of the most recently created dialog. To get the window handle to a dialog control, the GetItem method can be used.

4.9.8. GetSystemMetrics

```
>>-aBaseDialog~GetSystemMetrics(--index--)--------------------><
```

Obtains the system metric value for the given index.

Arguments:

The only argument is:

index

The numeric index of the system metric.

Good documentation on the system metrics is found in the MSDN Library under the GetSystemMetrics function. This documentation contains both the numeric value of the different indexes and information on what is returned for each index. The documentation is easy to found using a Google search of "GetSystemMetrics MSDN Library"

Return value:

The return value is dependent on the index queried. Consult the MSDN library for information on the returned values.

Note: The OS will return 0 if an invalid index is used. Hoever, the return value for some indexes is also 0. The programmer will need to determine from the context if 0 is an error return or not. In addition, ooDialog will return -1 for some invalid argument values. For instance, if the string "dog" was passed in for the `index` argument, -1 would be returned.

Example:

The following code snippet is from an application where the user can have 5, 6, or more, independent dialogs open at one time. One of the menu options is "Tile Dialogs." When the user selects this option all the open dialogs are "tiled."

All the open dialog objects are stored in a queue. In the `onTile` method, which is invoked when the user selects the menu item, each dialog is fetched in turn from the queue. Then the dialog is repositioned at an offset from the preceding dialog. It is shifted to the right the width of 2 scroll bars and shifted down the the width of the title bar. (This width is the title bar width plus the thin border around the title bar.) The height of the thin border, the height of the title bar, and the width of a scroll bar are all determined by querying the system metrics.

```
::method onTile
  expose offSetX offSetY dialogQueue

  -- SM_CXVSCROLL = 20
  -- SM_CYCAPTION = 4
  -- SM_CYBORDER  = 6

  if \ offSetX~datatype('W') then do
    scrollBarX = self~getSystemMetrics(20)
    titleBarY = self~getSystemMetrics(4)
    windowBorderY = self~getSystemMetrics(6)

    offSetX = 2 * scrollBarX
    offSetY = (2 * windowBorderY) + titleBarY
  end

  parse value self~getWindowRect(self~dlgHandle) with x y .

  do dlg over dialogQueue
    x = x + offSetX
    y = y + offSetY

    self~setWindowRect(dlg~dlgHandle, x, y, 0, 0, "NOSIZE")
  end
```

4.10. Appearance and Behavior Methods

The methods listed in this section are related to the appearance or the behavior of the dialog or its controls. The section contains methods related to size, position, visibility, and title.

Some of the methods come in two flavors, normal (for example, ShowWindow and fast (for example, ShowWindowFast). The fast extension indicates that the method does not redraw the control or window immediately. After modifying several items, invoke the Update method (see page Update) to redraw the dialog.

4.10.1. BackgroundColor

```
>>-aBaseDialog~BackgroundColor(--color--)--------------------><
```

The BackgroundColor method sets the background color of a dialog.

Arguments:

The only argument is:

color

A color-palette index specifying the background color. For information on the color numbers, refer to Definition of Terms.

Return value:

This method does not return a value.

4.10.2. Show

```
                        +-NORMAL---+
>>-aBaseDialog~Show(--"--+----------+--"--)--------------------><
                        +-DEFAULT--+
                        +-SHOWTOP--+
                        +-HIDE-----+
                        +-MIN------+
                        +-MAX------+
                        +-INACTIVE-+
```

The Show method shows the dialog. It is called by Execute or ExecuteAsync methods to initially display the dialog. Once the dialog has been initially shown, the programmer can use the `Show` to change the appearance of the dialog. For instance the dialog can be hidden, maximized, minimized, etc..

Argument:

The argument can be one of the following keywords, case is not significant:

NORMAL

Makes the dialog visible with its default window size. This is the default keyword if the argument is omitted.

DEFAULT

`DEFAULT` is an alias for `NORMAL`. The two keywords are functionally identical.

SHOWTOP

Shows and makes the dialog the topmost dialog.

HIDE

Makes the dialog invisible.

MIN

Minimizes the dialog.

MAX

Maximizes the dialog.

INACTIVE

Shows the dialog without changing the active window. When the `Normal` keyword is used, the dialog is shown and becomes the active window. Using the `INACTIVE` keyword allows the dialog to be shown and let another window remain active.

Example:

The following statement hides the dialog:

```
MyDialog~Show("HIDE")
```

4.10.3. ToTheTop

```
>>-aBaseDialog~ToTheTop----------------------------------------><
```

The ToTheTop method makes the dialog the topmost dialog.

Example:

The following example uses the ToTheTop method to make the user aware of an alarm event:

```
aDialog = .MyDialog~new
msg = .Message~new(aDialog, "Remind")
a = .Alarm~new("17:30:00", msg)
```

```
::class MyDialog subclass UserDialog
      .
      .
      .
::method Remind
  self~SetStaticText(102, "Don't forget to go home!")
  self~ToTheTop
```

Note: The Message and Alarm classes are built-in classes of Object Rexx. See the *Open Object Rexx: Reference* for further information.

4.10.4. EnsureVisible

```
>>-aBaseDialog~EnsureVisible----------------------------------><
```

The `EnsureVisible` method causes the dialog to reposition itself so that the entire dialog is on the visible screen. If the entire dialog is already on the visible screen then no action is taken.

This is useful in a number of situations. It allows the programmer to move or resize the dialog and not have to worry that the dialog is off the screen. After finishing the move or resizing, the programmer can invoke the EnsureVisible method and know that the dialog is entirely on the screen.

Arguments:

The method takes no arguments.

Return value:

This method always returns 0.

Example:

The following example reads in the previous size and position of the dialog and then opens the dialog in the position that the user had closed it. If the INI file is corrupted, or had been accidently edited, or a number of other things, it is possible that sizing and positioning the dialog using the values from the INI file will place the dialog of the screen. Invoking `EnsureVisible` will cause the dialog to reposition itself so that it is completely on the screen, but only if needed. If the dialog is already completely on the screen, then no action is taken.

```
if \.useDefault then
    do
        -- Read oorexxtry.ini position & size the dialog based on its values
        handle = self~getSelf()
        k1 = SysIni('oorexxtry.ini','oorexxtry','k1')
        k2 = SysIni('oorexxtry.ini','oorexxtry','k2')
        k3 = SysIni('oorexxtry.ini','oorexxtry','k3')
        k4 = SysIni('oorexxtry.ini','oorexxtry','k4')
        if k1 = 'ERROR:' | k2 = 'ERROR:' | k3 = 'ERROR:' | k4 = 'ERROR:' then
            nop -- First execution will not find the ini file
```

```
        else
            do
                self~setWindowRect(handle,k1,k2,k3-k1,k4-k2)
                self~ensureVisible
            end
    end
```

4.10.5. Minimize

```
>>-aBaseDialog~Minimize----------------------------------------><
```

The `Minimize` method minimizes the dialog to the taskbar. This is the identical to the user clicking the minimize button on the dialog window. This method will minimize the dialog even if it does not have the `MINIMIZEBOX` style when it is created.

This is a convenience method. It is functionally equivalent to using the Show method with the `MIN` keyword.

Arguments:

The method takes no arguments.

Return value:

0

Minimizing was successful.

1

Minimizing failed.

Example:

This example creates a secondary dialog to display the contents of a file. The secondary dialog is show full screen (maximized.) At the same time the parent dialog is minimized to the taskbar. The complete program listing is available (see the File Viewer example.)

```
::method onView
  expose viewDlg editCntrl

  fileName = editCntrl~getText
  viewDlg = .Viewer~new( , "fileView.h", self, fileName)
  if viewDlg~initCode = 0 then do
    self~disableItem(IDC_PB_VIEW)

    viewDlg~create(30, 30, 170, 180, "Viewer", "MAXIMIZEBOX MINIMIZEBOX")
    viewDlg~popUpAsChild(self, "HIDE", , IDI_DLG_APPICON)

    -- The underlying Windows dialog has to be created before it can be maximized.
```

```
    j = SysSleep(.1)

    viewDlg~maximize
    self~minimize
  end
```

4.10.6. Maximize

```
>>-aBaseDialog~Maximize----------------------------------------><
```

The `Maximize` method maximizes the dialog on the screen. This is the identical to the user clicking the maximize button on the dialog window. This method will maximize the dialog even if it does not have the `MAXIMIZEBOX` style when it is created.

This is a convenience method. It is functionally equivalent to using the Show method with the `MAX` keyword.

Arguments:

The method takes no arguments.

Return value:

0

Maximizing was successful.

1

Maximizing failed.

Example:

The following code snippet minimizes the dialog to the taskbar. It is used in the FileViewer example. The complete program listing is available (see the File Viewer example.)

```
::method onView
  expose viewDlg editCntrl

  ...

  if viewDlg~initCode = 0 then do
    ...

    viewDlg~maximize
    self~minimize
  end
```

4.10.7. Restore

```
>>-aBaseDialog~Restore----------------------------------------><
```

The `Restore` method restores a minimized or maximized dialog to its original position. This is the identical to the user taking action to restore the dialog window. The programmer can use this method to restore a dialog window that he previously minimized or maximized.

This is a convenience method. It is functionally equivalent to using the Show method with the `RESTORE` keyword.

Arguments:

The method takes no arguments.

Return value:

0

Restoring was successful.

1

Restoring failed.

Example:

The following example checks if the parent dialog is minimized and, if it is, the `Restore` method is used to show the dialog in the size and position it was prior to being minimized. The complete program listing for this code snippet is available (see the File Viewer example.)

```
::method cancel
  expose parent
  parent~enableItem(IDC_PB_VIEW)
  if parent~isMinimized then parent~restore
  return self~cancel:super
```

4.10.8. IsMinimized

```
>>-aBaseDialog~IsMinimized-------------------------------------><
```

The `IsMinimized` method is used to check if a dialog is currently minimized.

Arguments:

The method takes no arguments.

Return value:

.true

The dialog is minimized.

.false

The dialog is not minimized.

Example:

The following example checks if the parent dialog is minimized and, if it is, the `Restore` method is used to show the dialog in the size and position it was prior to being minimized. The complete program listing that this code snippet comes from is available (see the File Viewer example.)

```
::method cancel
  expose parent
  parent~enableItem(IDC_PB_VIEW)
  if parent~isMinimized then parent~restore
  return self~cancel:super
```

4.10.9. IsMaximized

```
>>-aBaseDialog~IsMaximized------------------------------------><
```

The `IsMaximized` method is used to check if a dialog is currently maximized.

Arguments:

The method takes no arguments.

Return value:

.true

The dialog is maximized.

.false

The dialog is not maximized.

Example:

The following code snippet would check if the dialog is maximized and, if it is, restore it to the position and size it was prior to being maximized.

```
  if self~isMaximized then self~restore
```

4.10.10. FocusItem

```
>>-aBaseDialog~FocusItem(--id--)------------------------------><
```

The FocusItem method sets the input focus to a particular dialog item.

Arguments:

The only argument is:

id

The ID of the dialog item to set the focus to

Return value:

This method always returns 1.

4.10.11. TabToNext

```
>>-aBaseDialog~TabToNext-------------------------------------><
```

The TabToNext method sets the input focus to the next dialog control with a tab stop. This performs the same action as if the user pressed the tab key in the dialog. Like many of the other methods, TabToNext is a method of both a BaseDialog and a DialogControl.

Return value:

-1

The method failed.

0

The focus was changed, but the control with the previous focus could not be determined.

Other

The handle to the control with the previous focus.

4.10.12. TabToPrevious

```
>>-aBaseDialog~TabToPrevious----------------------------------><
```

This method is the reverse of TabToNext and sets the input focus to the previous dialog control with a tab stop. This performs the same action as if the user pressed the shift-tab key in the dialog. TabToPrevious is a method of both a BaseDialog and a DialogControl.

Return value:

-1

The method failed.

0

The focus was changed, but the control with the previous focus could not be determined.

Other

The handle to the control with the previous focus.

4.10.13. SetGroup

```
>>-aBaseDialog~SetGroup(--id--,--+-----------+--)--------------><
                                 +-wantStyle-+
```

Add or remove the group style for the specified control. The group style controls how the user can navigate through the dialog using the keyboard. For most dialogs this does not change while the dialog is executing. However, in some dialogs the programmer may want to change the navigation depending on the options the user selects.

Arguments:

The arguments are:

id

The resource ID of the dialog control that will gain or lose the group style.

wantStyle

A boolean (.true or .false) to indicate whether the dialog control should have or not have the group style. True (the default) indicates the control should have the group style and false indicates the control should not have the style.

Return value:

Negative values indicate the function failed, non-negative values indicate success.

-4 or less

The value is the negated Operating System Error code. The absolute value of the return can be used to look up the error reason in the Windows documentation.

-3

The second argument to the method is not a boolean.

-2

There is an (internal) problem with the dialog or the dialog handle.

-1

The resource ID of the control is not correct.

0 or greater

The window style of the dialog control prior to adding or removing the group style.

4.10.14. SetTabStop

```
>>-aBaseDialog~SetTabStop(--id--,--+-----------+--)-------------><
                                   +-wantStyle-+
```

Add or remove the tab stop style for the specified control. When a control has the tabstop style, the user can set the focus to the control by using the tab key. When a control does not have this style, the tab key will skip over the control. Adding or removing this style during the execution of a dialog allows the programmer to alter how the user navigates through the dialog controls.

Arguments:

The arguments are:

id

The resource ID of the dialog control that will gain or lose the tabstop style.

wantStyle

A boolean (.true or .false) to indicate whether the dialog control should have or not have the tabstop style. True (the default) indicates the control should have the tabstop style and false indicates the control should not have the style.

Return value:

Negative values indicate the function failed, non-negative values indicate success.

-4 or less

The value is the negated Operating System Error code. The absolute value of the return can be used to look up the error reason in the Windows documentation.

-3

The second argument to the method is not a boolean.

-2

There is an (internal) problem with the dialog or the dialog handle.

-1

The resource ID of the control is not correct.

0 or greater

The window style of the dialog control prior to adding or removing the tabstop style.

4.10.15. EnableItem

```
>>-aBaseDialog~EnableItem(--id--)----------------------------><
```

The EnableItem method enables the given dialog item.

Arguments:

The only argument is:

id

The ID of the item

4.10.16. DisableItem

```
>>-aBaseDialog~DisableItem(--id--)----------------------------><
```

The DisableItem method disables the given dialog item. A disabled dialog item is usually indicated by a gray instead of a black title or text; it cannot be changed by the user.

Arguments:

The only argument is:

id

The ID of the item

4.10.17. HideItem

```
>>-aBaseDialog~HideItem(--id--)------------------------------><
```

The HideItem method makes the given item disappear from the screen and thus unavailable to the user. In fact, the item is still in the dialog and you can transfer its data.

Arguments:

The only argument is:

id

The ID of the item

4.10.18. HideItemFast

```
>>-aBaseDialog~HideItemFast(--id--)--------------------------><
```

The HideItemFast method hides an item without redrawing its area. It is similar to the HideItem method, but it is faster because the item's area is not redrawn. The HideItemFast method is used when more than one item state is modified. After the operations, you can manually redraw the dialog window, using the Update method.

Arguments:

The only argument is:

id

The ID of the item

4.10.19. ShowItem

```
>>-aBaseDialog~ShowItem(--id--)------------------------------><
```

The ShowItem method makes the given dialog item reappear on the screen.

Arguments:

The only argument is:

id

The ID of the item

4.10.20. ShowItemFast

```
>>-aBaseDialog~ShowItemFast(--id--)---------------------------><
```

The ShowItemFast method shows an item without redrawing its area. It is the counterpart to the HideItemFast method.

4.10.21. HideWindow

```
>>-aBaseDialog~HideWindow(--hwnd--)---------------------------><
```

The HideWindow method hides a whole dialog window or a dialog item.

Arguments:

The only argument is:

hwnd

A handle to the window or dialog item. Use the GetSelf or GetItem method to get a handle.

Example:

The following example gets the window handle of the top dialog (which is not necessarily the executing dialog, see the Get method) and hides the whole dialog:

```
hwnd = MyDialog~Get
MyDialog~HideWindow(hwnd)
```

4.10.22. HideWindowFast

```
>>-aBaseDialog~HideWindowFast(--hwnd--)-----------------------><
```

The HideWindowFast method is similar to the HideWindow method, but it is faster because the window's or item's area is not redrawn. The HideWindowFast method is used when more than one state is modified. After the operations, you can manually redraw the dialog window, using the Update method.

Arguments:

The only argument is:

hwnd

A handle to the window or dialog item

4.10.23. ShowWindow

```
>>-aBaseDialog~ShowWindow(--hwnd--)--------------------------><
```

The ShowWindow method shows the window or item again.

Arguments:

The only argument is:

hwnd

The handle of a window or an item

4.10.24. ShowWindowFast

```
>>-aBaseDialog~ShowWindowFast(--hwnd--)----------------------><
```

The ShowWindowFast method is the counterpart to the HideWindowFast method.

4.10.25. SetWindowRect

```
>>-aBaseDialog~SetWindowRect(--hwnd--,--x--,--y--,--width--,--height-->

>--+-----------------------------+--)-------------------------------><
   |       +----------------+    |
   |       V                |    |
   +-,--"----+-NOMOVE-----+-+--"-+
             +-NOSIZE-----+
             +-HIDEWINDOW-+
             +-SHOWWINDOW-+
             +-NOREDRAW---+
```

The SetWindowRect method sets new coordinates for a specific window.

Arguments:

The arguments are:

hwnd

The handle to the dialog that is to be repositioned.

x, y

The new position of the upper left corner, in screen pixels.

width

The new width of the window, in screen pixels.

height

The new height of the window, in screen pixels.

showOptions

This argument can be one or more of the following keywords, separated by blanks:

NOMOVE

The upper left position of the window has not changed.

NOSIZE

The size of the window has not changed.

HIDEWINDOW

The window is to be made invisible.

SHOWWINDOW

The window is to be made visible.

NOREDRAW

The window is to be repositioned without redrawing it.

Return value:

0

Repositioning was successful.

1

Repositioning failed.

4.10.26. RedrawWindow

```
>>-aBaseDialog~RedrawWindow(--hwnd--)------------------------><
```

The RedrawWindow method redraws a specific dialog.

Arguments:

The only argument is:

hwnd

The handle to the dialog that is to be redrawn.

Return value:

0

Redrawing was successful.

1

Redrawing failed.

4.10.27. ResizeItem

```
>>-aBaseDialog~ResizeItem(--id--,--width--,--height--+-------------------------+--)-><
                                                     +-,--"--+-HIDEWINDOW-+--"-+
                                                             +-SHOWWINDOW-+
                                                             +-NOREDRAW---+
```

The ResizeItem method changes the size of a dialog item.

Arguments:

The arguments are:

id

The ID of the dialog item you want to resize

width, height

The new size in dialog units

showOptions

This argument can be one of the following keywords:

HIDEWINDOW

Hides the item

SHOWWINDOW

Shows the item

NOREDRAW

Resizes the item without updating the display. Use the Update method to manually update the display.

Example:

The following example resizes a dialog item:

```
MyDialog~ResizeItem(123, 40, 30, "SHOWWINDOW")
```

4.10.28. MoveItem

```
>>-aBaseDialog~MoveItem(--id--,--xPos--,--yPos--+-------------------------+--)-><
                                                +-,--"--+-HIDEWINDOW-+--"-+
                                                        +-SHOWWINDOW-+
                                                        +-NOREDRAW---+
```

The MoveItem method moves a dialog item to another position within the dialog window.

Arguments:

The arguments are:

id

The ID of the dialog item you want to move

xPos, yPos

The new position in dialog units relative to the dialog window

showOptions

This argument can be one of the following keywords:

HIDEWINDOW

Hides the dialog

SHOWWINDOW

Shows the dialog

NOREDRAW

Moves the dialog item without updating the display. Use the Update method to manually update the display.

4.10.29. Center

```
>>-aBaseDialog~Center(--"--+-HIDEWINDOW-+--"--)---------------><
                          +-SHOWWINDOW-+
                          +-NOREDRAW---+
```

The Center method moves the dialog to the screen center.

Arguments:

The only argument can be one of:

HIDEWINDOW

Hides the dialog

SHOWWINDOW

Shows the dialog

NOREDRAW

Center the dialog without updating the display. Use the Update method to manually update the display.

4.10.30. SetWindowTitle

```
>>-aBaseDialog~SetWindowTitle(--hwnd--,--aString--)-----------><
```

The SetWindowTitle method changes the title of a window.

Arguments:

The arguments are:

hwnd

The handle of the window whose title you want to change

aString

The new title text

4.10.31. FileViewer Example Program

The FileViewer program is a complete working program that uses many of the methods of the base dialog. Portions of this program are presented as examples in the documentation for these methods. The complete program is shown here as a reference to how the code snippets all fit together.

The documentation for the individual methods used in the program has additional commentary that will help in understanding how the program works.

- Init: This program uses a header file, `fileView.h` to define symbolic IDs for the dialog controls. The name of the file is one of the arguments used to create a new instance of a dialog object. (As in all ooRexx classes, the arguments used in `new` are passed on to the `init` method of the class.)
- Execute: Two dialogs are created in the example. The first dialog allows the user to enter the name of a file to view. Then the file itself is displayed in a second dialog. Each dialog uses a different application icon. The `Execute` method documentation has more detail on the application icon.
- InitDialog This method is called after the underlying Windows dialog has been created, but before it is shown on the screen. In the viewer dialog the `Init` method is overridden and the text of the read-only multi-line edit control is set to the contents of the file. In both dialogs, this method is used to get and save a reference to the edit control of the dialog.
- DisableItem: When the user clicks the "View File" button, the button is disabled until the user is finished viewing the file. When a button is disabled it can not be clicked by the user.
- EnableItem: When the user closes the secondary viewer dialog, the "View File" button is enabled so the user can choose to view another file.
- PopupAsChild: The viewer dialog is executed using this method so that both dialogs run independently, both dialogs stay enabled. The `HIDE` keyword is used for the show argument. This creates the dialog as invisible and prevents unnecessary screen flicker.
- Maximize: Note that there is a 100 millisecond sleep right before the `Maximize method` is invoked. This allows Windows to finish creating the dialog. The `InitDialog` method will have already finished executing and the contents of the file are loaded into the multi-line edit control. Then, `Maximize` resizes the dialog to take up the whole screen and shows it. The dialog appears on the user's screen, with the file displayed, in one screen drawing. This reduces the flicker on the screen.
- Minimize: When the viewer dialog is maximized, the main dialog is minimized to the task bar.
- InitAutoDetection: Because the contents of the file are loaded into the multi-line edit control during `InitDialog`, auto detection must be turned off. Otherwise, after `InitDialog` finishes executing, auto detection would set the empty string as the text for the control. The program overrides this method to turn auto detection off.
- NoAutoDetection: Used in turning auto detection off.
- ConnectResize: When the viewer dialog is resized, the `onSize` method is invoked. The size of the multi-line edit control is then changed so it completely takes up the client area of the dialog. (The client area of a window is where all the child windows are drawn. In this case the edit control is the only child window.) Since the dialog is not resizeable by the user (it does not have a sizing border) the only time the size can change is when the dialog is maximized, minimized, or restored.
- HideItem: The edit control in the viewer dialog is created invisible. Again, this helps reduce flicker.
- GetItem: The window handle of the edit control is obtained by this method ...
- GetClientRect: ... then the size of the client area of the viewer dialog is obtained using this method. Then ..
- SetWindowRect: ... the size of the edit control window is set using to the size of the dialog's client area.

- ShowItem: The `ShowItem` method is used to make the edit control visible when the dialog is initially shown.
- IsMinimized: When the viewer dialog is closed, this method checks to see if the main dialog is still minimized. Since the dialogs run independently the user may have already restored this dialog.
- Restore: If the main dialog is still minimized, then this method restores it to its normal position.

```
/* fileView.h  Simple symbolic ID definitions  */

#define IDD_DIALOG1          100
#define IDC_ST_TYPE          105
#define IDC_ENTRYLINE        106
#define IDC_MULTILINE        107
#define IDC_PB_VIEW          111

/* FileViewer.rex  Simple Dialog to view files full screen */

  dlg = .FileView~new( , "fileView.h")
  if dlg~initCode = 0 then do
    dlg~createCenter(170, 90, "The File Viewer Dialog", "VISIBLE MAXIMIZEBOX MINIMIZEBOX")
    dlg~Execute("SHOWTOP", IDI_DLG_OOREXX)
    dlg~Deinstall
  end

-- End of entry point.
::requires "OODWin32.cls"

::class FileView subclass UserDialog inherit AdvancedControls

::method defineDialog

  self~addText(10, 25, 150, 10, " Enter the name of a file to view:", "", IDC_ST_TYPE)
  self~addEntryLine(IDC_ENTRYLINE, , 10, 35, 150, 10, "AUTOSCROLLH")

  -- When the view button is pushed, another dialog will show the file.
  self~addButton(IDC_PB_VIEW, 10, 55, 35, 15, "View File", onView, "DEFAULT GROUP")
  self~addButton(IDOK, 130, 55, 35, 15, "Quit")

::method initDialog
  expose editCntrl
  editCntrl = self~getEditControl(IDC_ENTRYLINE)

::method onView
  expose viewDlg editCntrl

  fileName = editCntrl~getText
  viewDlg = .Viewer~new( , "fileView.h", self, fileName)
  if viewDlg~initCode = 0 then do
    self~disableItem(IDC_PB_VIEW)

    viewDlg~create(30, 30, 170, 180, "Viewer", "MAXIMIZEBOX MINIMIZEBOX")
```

```
    viewDlg~popUpAsChild(self, "HIDE", , IDI_DLG_APPICON)

    -- The underlying Windows dialog has to be created before it can be maximized.
    j = SleepMS(100)

    viewDlg~maximize
    self~minimize
  end

::class 'Viewer' subclass UserDialog inherit AdvancedControls

::method init
  expose parent filename
  use arg data, header, parent, fileName
  forward class (super)

::method initAutoDetection
  self~noAutoDetection

::method defineDialog
  expose wasMinimized

  wasMinimized = .false
  style = "VSCROLL HSCROLL MULTILINE READONLY"
  self~addEntryLine(IDC_MULTILINE, "cEntry", 0, 0, 170, 180, style)
  self~connectResize("onSize")

::method initDialog
  expose editControl fileName isHidden

  self~hideItem(IDC_MULTILINE)
  isHidden = .true

  editControl = self~getEditControl(IDC_MULTILINE)
  fObj = .stream~new(fileName)
  text = fObj~charin(1, fObj~chars)
  fObj~close
  if text == "" then text = "   No file  "
  editControl~setText(text)

::method onSize
  expose wasMinimized
  use arg sizeEvent sizeInfo

  if sizeEvent = 1 then wasMinimized = .true

  if sizeEvent = 0 |  sizeEvent = 2 then do
    if \ wasMinimized then self~resizeEditControl
    wasMinimized = .false
  end

::method resizeEditControl
  expose editControl isHidden
```

```
  hWnd = self~getItem(IDC_MULTILINE)
  parse value self~getClientRect with wx wy wcx wcy

  self~setWindowRect(hWnd, 0, 0, wcx, wcy)

  if isHidden then do
    self~showItem(IDC_MULTIINE)
    isHidden = .false
  end

::method cancel
  expose parent
  parent~enableItem(IDC_PB_VIEW)
  if parent~isMinimized then parent~restore
  return self~cancel:super
```

4.11. Window Draw Methods

The methods listed below are used to draw, redraw, and clear window areas.

4.11.1. DrawButton

```
>>-aBaseDialog~DrawButton(--id--)----------------------------><
```

The DrawButton method draws the given button.

Arguments:

The only argument is:

id

The ID of the button

4.11.2. RedrawRect

```
                                                                          +-0--------
+
>>-aBaseDialog~RedrawRect(--+------+--,--left--,--top--,--right--,--bottom--,--+----------
+--)-><
                            +-hwnd-+                                      +-erasebkg-
+
```

The RedrawRect method immediately redraws a rectangle within the client area of a dialog. You can specify whether the background of the dialog is to be erased before redrawing.

Arguments:

The arguments are:

hwnd

The handle to the dialog in which parts of the client area are to be redrawn. See Get or GetItem for information on how to get a window handle. If you omit this argument, the handle of the dialog instance is used.

left, top

The upper left corner of the rectangle relative to the client area, in screen pixels.

right, bottom

The lower right corner of the rectangle relative to the client area, in screen pixels.

erasebkg

If this argument is 1 or 0, the background of the dialog is erased before redrawing. The default is 0.

Return value:

0

Redrawing was successful.

1

Redrawing failed.

4.11.3. RedrawButton

```
>>-aBaseDialog~RedrawButton(--id--+-------------+--)---------><
                                  +-,--erasebkg-+
```

The RedrawButton method redraws the given button.

Arguments:

The arguments are:

id

The ID of the button

erasebkg

Determines whether (1) or not (0) the background of the drawing area should be erased before redrawing. The default is 0.

4.11.4. RedrawWindowRect

```
>>-aBaseDialog~RedrawWindowRect--+-----------------------------+-->
                                 +-(--hwnd--+-------------+--)-+
                                            +-,--erasebkg-+
```

The RedrawWindowRect method redraws the given window rectangle.

Arguments:

The arguments are:

hwnd

The handle to the window. See Get or GetItem for information on how to get a window handle. If you omit this argument, the handle of the dialog instance is used.

erasebkg

Determines whether (1) or not (0) the background of the drawing area should be erased before redrawing. The default is 0.

4.11.5. ClearRect

```
>>-aBaseDialog~ClearRect(--hwnd--,--left--,--top--,--right--,--bottom--)-->
```

The ClearRect method clears the given rectangle of a window. The values are in pixels.

Arguments:

The arguments are:

hwnd

The handle of the window. See GetSelf or GetItem for information on obtaining a window handle.

left

The horizontal value of the upper-left corner of the rectangle

top

The vertical value of the upper left corner

right

The horizontal value of the lower right corner

bottom

The vertical value of the lower right corner

Example:

The following example clears a rectangle of the size 20 by 20:

```
hwnd=MyDialog~GetSelf
MyDialog~ClearRect(hwnd, 2, 4, 22, 24)
```

4.11.6. ClearButtonRect

```
>>-aBaseDialog~ClearButtonRect(--id--)------------------------><
```

The ClearButtonRect method erases the draw area of the given button.

Arguments:

The only argument is:

id

The ID of the push button

4.11.7. ClearWindowRect

```
>>-aBaseDialog~ClearWindowRect(--hwnd--)----------------------><
```

The ClearWindowRect method erases the draw area of the given window.

Arguments:

The only argument is:

hwnd

The handle of the window. See GetSelf, Get, or GetItem for some methods to get a window handle.

Example:

The following example gets the window handle of a dialog control with resource ID of 211 and then clears the dialog control window:

```
hwnd = MyDialog~GetItem(211)
```

```
MyDialog~ClearWindowRect(hwnd)
```

4.12. Bitmap Methods

The methods listed below deal with bitmaps.

4.12.1. ChangeBitmapButton

```
>>-aBaseDialog~ChangeBitmapButton(--id--,--bmpNormal----------------------->

>--+-------------------------------------------------------------------+-->
   +-,bmpFocused--+-------------------------------------------------+-+
                  +-,bmpSelected--+--------------------------------+-+
                                  +-,bmpDisabled--+---------------+-+
                                                  +-,styleOptions-+

>--)----------------------------------------------------------------------><
```

The ChangeBitmapButton method changes the bitmaps of a bitmap button.

Arguments:

The arguments are the same as for ConnectBitmapButton, except for the first argument (msgToRaise), which is skipped in this method.

Example:

The following example replaces the current bitmap with a new bitmap:

```
MyDialog~ChangeBitmapButton(501, "NewBB.bmp")
```

4.12.2. GetBitmapSizeX

```
>>-aBaseDialog~GetBitmapSizeX(--id--)------------------------><
```

The GetBitmapSizeX method returns the horizontal bitmap extension.

Arguments:

The only argument is:

id

The ID of the bitmap button

4.12.3. GetBitmapSizeY

```
>>-aBaseDialog~GetBitmapSizeY(--id--)------------------------><
```

The GetBitmapSizeY method returns the vertical bitmap extension.

Arguments:

The only argument is:

id

The ID of the bitmap button

4.12.4. DrawBitmap

```
>>-aBaseDialog~DrawBitmap(--+------+--,--id---------------------->
                            +-hwnd-+

>--+-------------------------------------------------------------+-->
   +-,--px--,--py--+---------------------------------------+-+
                   +-,--srcx--,--srcy--+------------------+-+
                                       +-,--xlen--,--ylen-+

>--)-------------------------------------------------------------><
```

The DrawBitmap method draws the bitmap of a button. You can also use this method to move a bitmap or a part of it.

Arguments:

The arguments are:

hwnd

The handle to the window. If this argument is omitted, the handle for the button is used automatically.

id

The ID of the button that has the owner-draw option set

px, py

The upper-left corner of the target space within the button (default is 0)

srcx, srcy

The upper-left corner within the bitmap (default is 0)

xlen, yLen

The extension of the bitmap or a part of it (default is the whole bitmap)

4.12.5. ScrollBitmapFromTo

```
>>-aBaseDialog~ScrollBitmapFromTo(--id--,--fromX--,--fromY--,--toX--,--toY--,-->

>--stepX--,--stepY--,--delay--,--displace--)------------------><
```

The ScrollBitmapFromTo method scrolls a bitmap from one position to another within an owner-drawn button.

Arguments:

The arguments are:

id

The ID of the button

fromX, fromY

The starting position

toX, toY

The target position

stepX, stepY

The width of one step

delay

The time in milliseconds this method waits after each move before doing the next move. This determines the speed at which the bitmap moves.

displace

If set to 1 the internal position of the bitmap (bitmap displacement) is updated after each incremental move. DisplaceBitmap is called after each step to adjust the bitmap position. If the dialog is redrawn, the bitmap is shown at the correct position, but the drawing is slower.

4.12.6. TiledBackgroundBitmap

```
>>-aBaseDialog~TiledBackgroundBitmap(--bmpFilename--)---------><
```

The TiledBackgroundBitmap method sets a bitmap as the background brush (Windows NT® only). If the bitmap size is less than the size of the background, the bitmap is drawn repetitively.

Arguments:

The only argument is:

bmpFilename

The name of a bitmap file

4.12.7. BackgroundBitmap

```
>>-aBaseDialog~BackgroundBitmap(--bmpFilename------------------->

>--+-------------+--)----------------------------------------><
   +-,--"USEPAL"-+
```

The BackgroundBitmap method sets a bitmap as the dialog's background picture.

Arguments:

The arguments are:

bmpFilename

The name of a bitmap file

option

Set the last argument to USEPAL if you want to use the color palette of the bitmap. See ConnectBitmapButton for more information.

4.12.8. DisplaceBitmap

```
>>-aBaseDialog~DisplaceBitmap(--id--,--x--,--y--)------------><
```

The DisplaceBitmap method sets the position of a bitmap within a button.

Arguments:

The arguments are:

id

The ID of a button

x

The horizontal displacement in screen pixels. A negative value can be used.

y

The vertical displacement (negative allowed)

Example:

The following example moves the bitmap within a button four screen pixels to the right and three pixels upward:

```
MyBaseDialog~DisplaceBitmap(244, 4, -3)
```

4.12.9. GetBmpDisplacement

```
>>-aBaseDialog~GetBmpDisplacement(--id--)--------------------><
```

The GetBmpDisplacement method gets the position of a bitmap within a button.

Arguments:

The only argument is:

id

The ID of the button

Example:

The following example shows how to use the GetButtonRect and GetBmpDisplacement methods:

```
bRect = MyBaseDialog~GetButtonRect(244)
parse var bRect left top right bottom
bmpPos = MyBaseDialog~GetBmpDisplacement(244)
parse var bmpPos x y
```

4.13. Device Context Methods

The methods listed below are used to retrieve and release a device context.

A device context is associated with a window, a dialog, or a push button, and is a drawing area managed by a window. A device context stores information about the graphic objects (bitmaps, lines, pixels, ...) that are displayed and the tools (pen, brush, font, ...) that are used to display them.

4.13.1. GetWindowDC

```
>>-aBaseDialog~GetWindowDC(--hwnd--)-------------------------><
```

The GetWindowDC method returns the device context of a window. Do not forget to free the device context after you have completed the operations (see FreeWindowDC).

Arguments:

The only argument is:

hwnd

The handle of the window

4.13.2. GetButtonDC

```
>>-aBaseDialog~GetButtonDC(--id--)----------------------------><
```

The GetButtonDC method returns the device context of a button. Do not forget to free the device context after you have completed the operations (see FreeButtonDC).

Arguments:

The only argument is:

id

The ID of the button

4.13.3. FreeWindowDC

```
>>-aBaseDialog~FreeWindowDC(--hwnd--,--dc--)------------------><
```

The FreeWindowDC method frees the device context of a window.

Arguments:

The arguments are:

hwnd

The window handle

dc

The device context previously received by the GetWindowDC method

4.13.4. FreeButtonDC

```
>>-aBaseDialog~FreeButtonDC(--id--,--dc--)-------------------><
```

The FreeButtonDC method releases the device context of a button.

Arguments:

The arguments are:

id

The ID of the button

dc

The device context previously received by the GetButtonDC method

4.14. Text Methods

The following methods are used to display text dynamically in a window area and to modify the state of a device context. See GetWindowDC, GetDC, and GetButtonDC for information on how to retrieve a device context.

4.14.1. Write

```
>>-aBaseDialog~Write(--xPos--,--yPos--,--text---------------------------------->

>--+-----------------------------------------------------------------------+-->
   |    +-"SYSTEM"-+                                                        |
   +-,--+----------+--+-----------------------------------------------------+-+
        +-fontName-+  |    +-10-------+                                     |
                      +-,--+----------+--+-----------------------------+-+
                           +-fontSize-+  |       +-----------------+    |
                                         |       V +-OPAQUE------+ |    |
                                         +-,--"----+-THIN--------+-+--"-+
                                                   +-EXTRALIGHT--+
                                                   +-LIGHT-------+
                                                   +-MEDIUM------+
                                                   +-SEMIBOLD----+
                                                   +-EXTRABOLD---+
                                                   +-BOLD--------+
```

```
                                                        +-HEAVY-------+
                                                        +-UNDERLINE---+
                                                        +-ITALIC------+
                                                        +-STRIKEOUT---+
                                                        +-TRANSPARENT-+
                                                        +-CLIENT------+

>--)-----------------------------------------------------------------------><
```

The Write method enables you to write text to the dialog in the given font and size, to the given position. This method does not take a handle or an ID; it always writes to the dialog window.

Arguments:

See WriteToWindow for a description of the other arguments.

4.14.2. ScrollText

```
>>-aBaseDialog~ScrollText(--hwnd--,--text--,--+----------+--,--+----------+-->
                                              +-fontName-+     +-fontSize-+

>--,--+-------------------------+------------------------------------------>
      |    +----------------+    |
      |    V                |    |
      +-"----+-THIN-------+-+--"-+
             +-EXTRALIGHT-+
             +-LIGHT------+
             +-MEDIUM-----+
             +-SEMIBOLD---+
             +-EXTRABOLD--+
             +-BOLD-------+
             +-HEAVY------+
             +-UNDERLINE--+
             +-ITALIC-----+
             +-STRIKEOUT--+

>--,--displaceY--,--step--,--sleep--,--color--)----------------------------><
```

The ScrollText method scrolls text in a window with the given size, font, and color. The text is scrolled from right to left. If the method is started concurrently, call it a second time to stop scrolling.

Arguments:

The arguments are:

hwnd

The handle of the window in which the text is scrolled

text

A text string that is scrolled

displaceY

The vertical displacement of the text relative to the top of the window's client area (default 0)

step

The size of one step in screen pixels (default 4)

sleep

The time in milliseconds that the program waits after each movement (default 10). This determines the speed.

color

The color of the text (default 0, black)

See WriteToWindow for a description of the other arguments.

Example:

The following example scrolls the string "Hello world!" from left to right within the given window. The text is located two pixels below the top of the client area, one move is 3 screen pixels, and the delay time after each movement is 15 ms.

```
MyDialog~ScrollText(hwnd, "Hello world!", , , , 2, 3, 15)
```

Note: Only one sleep interval can be set for multiple scrolling texts within one process. All scrolling text in one process is synchronized with the first given interval.

4.14.3. ScrollInButton

```
>>-aBaseDialog~ScrollInButton(--id--,--text--,--+----------+--,-->
                                                +-fontName-+

>--+----------+--,--+-------------------------+---------------->
   +-fontSize-+     |    +----------------+    |
                    |    V                |    |
                    +-"----+-THIN-------+-+--"-+
                           +-EXTRALIGHT-+
                           +-LIGHT------+
                           +-MEDIUM-----+
                           +-SEMIBOLD---+
                           +-EXTRABOLD--+
                           +-BOLD-------+
                           +-HEAVY------+
                           +-UNDERLINE--+
                           +-ITALIC-----+
                           +-STRIKEOUT--+
```

```
>--,--displaceY--,--step--,--sleep--,--color--)--------------><
```

The ScrollInButton method scrolls text within a button. It is similar to the ScrollText method, except that you have to pass an ID instead of a window handle.

4.14.4. ScrollButton

```
>>-aBaseDialog~ScrollButton(--id--,--xPos--,--yPos--,--left--,--top-->

>--,--right--,--bottom--)-----------------------------------><
```

The ScrollButton method moves the rectangle within a button. It is used to move bitmaps within buttons.

Arguments:

The arguments are:

id

The ID of the button

xPos, yPos

The new position of the rectangle (in pixels)

left, top, right, bottom

The extension of the rectangle

4.14.5. SetItemFont

```
                                                  +-1--------+
>>--aBaseDialog~SetItemFont(--id--,--fonthandle--,--+----------+--)--><
                                                  +--redraw--+
```

The SetItemFont method changes the font for a particular dialog item.

The best place to call this method is within InitDialog. If the font is no longer needed, for instance, when the dialog is closed or another font has been assigned to the dialog item, you should free the font resource by calling DeleteFont. A good place to do this is the Leaving method.

Arguments:

The arguments are:

id

The ID of the dialog item.

fonthandle

The handle returned by CreateFont.

redraw

0

Do not redraw the item.

1

Redraw the item, which is the default.

Example:

The following example sets a 12-point Arial font for item 101.

```
::method InitDialog
      .
      .
      .
   hFont=self~CreateFont("Arial",12)
   self~SetItemFont(101,hFont,0)
```

4.15. Animated Buttons

The methods listed below work with animated buttons.

4.15.1. AddAutoStartMethod

```
>>-aBaseDialog~AddAutoStartMethod(--+---------+--,--MethodName--+---------------+--)-><
                                   +-InClass-+                 +-,--Parameters-+
```

The AddAutoStartMethod method adds a method name and parameters to a special internal queue. All methods in this queue will be started automatically and run concurrently when the dialog is executed. The given method (MethodName) in the given class (InClass) is started concurrently with the dialog when the dialog is activated using the Execute or ExecuteAsync method. This is useful for processing animated buttons.

Arguments:

The arguments are:

InClass

The class where the method is defined. If this argument is omitted, the method is assumed to be defined in the dialog class.

MethodName

The name of the method

Parameters

All parameters that are passed to this method

Example:

The following example installs the ExecuteB method of the MyAnimatedButton class so that it is processed concurrently with the dialog execution:

```
MyDialog~AddAutoStartMethod("MyAnimatedButton", "ExecuteB")

::class MyAnimatedButton
::method ExecuteB
      .
      .
      .
```

4.15.2. ConnectAnimatedButton

```
>>-aBaseDialog~ConnectAnimatedButton(--id,--+------------+--,--->
                                            +-msgToRaise-+

>--+-----------+--,--bmpFrom--,--+-------+--,--moveX--,--moveY--,-->
   +-AutoClass-+                 +-bmpTo-+

>--+-------+--,--+-------+--,--delay--,--+------+--,--+------+--)-><
   +-sizeX-+     +-sizeY-+               +-xNow-+     +-yNow-+
```

The ConnectAnimatedButton method installs an animated button and runs it concurrently with the main activity.

Arguments:

The arguments are:

id

The ID of the button

msgToRaise

The name of a method within the same class. This method is called each time the button is clicked.

AutoClass

The class that controls the animation (default is AnimatedButton Class)

bmpFrom

The ID of the first bitmap in the animation sequence within a binary resource. It can also be the name of an array stored in the .local directory containing handles of bitmaps to be animated and bmpTo is omitted. See LoadBitmap for how to get bitmap handles. The array starts at index 1.

bmpTo

The ID of the last bitmap in the animation sequence within a binary resource. If omitted, bmpFrom is expected to be the name of an array stored in .local that holds the bitmap handles of the bitmaps that are to be animated.

moveX, moveY

Size of one move (in pixels)

sizeX, sizeY

Size of the bitmaps (if omitted, the size of the bitmaps is retrieved)

delay

The time in milliseconds the method waits after each move

xnow, ynow

The starting position of the bitmap

Example:

The following example defines and runs an animated button. The example loads ten bitmaps ("anibmp1.bmp" to "anibmp10.bmp") into memory and stores them into the array "My.Bitmaps" that is stored in the .local directory. The name "My.Bitmaps" is specified as the `bmpfrom` and `bmpto` is omitted. After the dialog execution the bitmaps are removed from memory again. The sample also uses a different animation class (".MyAnimation") which subclasses from .AnimatedButton and overrides method HitRight which plays a tune each time the animated bitmap hits the right border.

```
/* store array in .local */
.Local["My.Bitmaps"] = .array~new(10)
/* load 10 bitmaps into .local array */
do i= 1 to 10
  .Local["My.Bitmaps"][i] = Dialog~LoadBitmap("anibmp"i".bmp")
  /* you could also use .My.Bitmaps[i] =  ... */
end

/* connect bitmap sequence and .MyAnimated class with button IDANI */
```

```
Dialog~ConnectAnimatedButton("IDANI", ,.MyAnimation,"My.Bitmaps", ,1,1, , ,100)

...
Dialog~Execute
...

/* Free the bitmap previously loaded */
do bmp over .Local["My.Bitmaps"] /* You could also use do bmp over .My.Bitmaps */
   Dialog~RemoveBitmap(bmp)
end


::class MyAnimation subclass AnimatedButton

/* play sound.wav whenever the bitmap hits the right border */

::method HitRight
   ret = Play("sound.wav", yes)
   return self~super:hitright
```

4.16. Menu Methods

The methods listed below manipulate a menu connected to the dialog.

4.16.1. ConnectMenuItem

```
>>-aBaseDialog~ConnectMenuItem(--id--,--msgToRaise--)--------->
```

The ConnectMenuItem method is called to connect a menu item selection with a method.

Arguments:

The arguments are:

id

The ID of the menu item.

msgToRaise

The name of the method that is to be called.

Example:

See ConnectButton.

Do not use one of the menu item methods below, prior to the SetMenu method. If you call one of these methods before SetMenu has been called, the intended action will not be processed and the return code is unpredictable.

4.16.2. EnableMenuItem

```
>>-aBaseDialog~EnableMenuItem(--id--)------------------------><
```

The EnableMenuItem method is called to enable a menu item.

Arguments:

The only argument is:

id

The ID of the menu item to be enabled.

4.16.3. DisableMenuItem

```
>>-aBaseDialog~DisableMenuItem(--id--)-----------------------><
```

The DisableMenuItem method is called to disable a menu item.

Arguments:

The only argument is:

id

The ID of the menu item to be disabled.

4.16.4. CheckMenuItem

```
>>-aBaseDialog~CheckMenuItem(--id--)--------------------------><
```

The CheckMenuItem method is called to set the check mark for a menu item.

Arguments:

The only argument is:

id

The ID of the menu item to be checked.

4.16.5. UncheckMenuItem

```
>>-aBaseDialog~UncheckMenuItem(--id--)------------------------><
```

The UncheckMenuItem method is called to remove the check mark from a menu item.

Arguments:

The only argument is:

id

The ID of the menu item to be unchecked.

4.16.6. GrayMenuItem

```
>>-aBaseDialog~GrayMenuItem(--id--)---------------------------><
```

The GrayMenuItem method is called to disable a menu item. The menu item is grayed. EnableMenuItem is used to reset the grayed state.

Arguments:

The only argument is:

id

The ID of the menu item to be grayed.

4.16.7. SetMenuItemRadio

```
>>-aBaseDialog~SetMenuItemRadio(--idstart--,--idend--,--idset--)><
```

The SetMenuItemRadio method is used to change the selection for a radio button menu group.

Arguments:

The arguments are:

idstart

The ID of the first menu item in the group.

idend

The ID of the last menu item in the group.

idset

The ID of the menu item that is to be selected.

Example:

The following example shows how to change the selection within a radio button group. Menu item 102 gets the radio button.

```
self~SetMenuItemRadio(101, 105, 102)
```

4.16.8. GetMenuItemState

```
>>-aBaseDialog~GetMenuItemState(--id--)----------------------><
```

The GetMenuItemState method returns the state of a given menu item

Arguments:

The only argument is:

id

The ID of the menu item whose state is of interest.

Return values:

CHECKED DISABLED GRAYED HIGHLIGHTED

4.17. Debugging Method

The following method displays the internal setting of the dialog administration table.

4.17.1. Dump

The Dump method displays the internal settings of the dialog administration table. This method can be helpful for debugging OODialog programs.

```
>>-Dump(--+-------------+--)---------------------------------><
          +-dialogadmin-+
```

Argument:

The only argument is:

dialogadmin

A pointer to a particular dialog administration record. If you specify this argument, you get detailed information on this record. If you omit this argument, all administration records (one for each dialog of the active process) are listed.

Return value:

This method does not return a value.

Chapter 5. DialogControl Class

The DialogControl class provides methods that are common to all dialog controls. It is a generic class that serves as a superclass to all dialog control specific classes. In the graphical user interface (GUI) for the Windows operating system both dialogs and dialog controls are windows. Therefore, many of the methods of the DialogControl class are the same as the methods of the ooDialog dialog classes. These are the methods that are common to all windows, whether they are dialog windows or dialog control windows.

In the Windows GUI all windows are created with a set of *window styles*. Dialog control windows are created using styles that are common to all windows, (for example the visible style,) and styles specific to the dialog control itself, (for example the multi-line style of the EditControl class.) In general the styles of a control fall into three categories: Styles that can only be set when the control is created and then can not be changed afterwards. Styles that can be changed after the control is created by sending messages to the control. And, styles that can be changed after the control is created by accessing the control window directly.

The ooDialog programmer chooses the window styles for dialog controls when he defines the dialog, either by using the Add... Methods for a user dialog, or by using a resource editor for dialogs defined with resource scripts or binary compiled resources. After the dialog control has been created, the individual dialog control classes provide methods to change those styles that can be changed, either by sending the proper message to the control or by accessing the control window directly.

Requires:

The DialogControl class requires the class definition file `oodwin32.cls`:

```
::requires oodwin32.cls
```

Attributes:

Instances of the DialogControl class have the following attributes:

FactorX

The horizontal size of one dialog unit, in pixels.

FactorY

The vertical size of one dialog unit, in pixels.

SizeX

The width of the dialog control, in dialog units, when the ooRexx object that represents the control is created.

Note: The `SizeX` attribute is not changed by ooDialog after the object is instantiated. If the programmer resizes the dialog control manually, and requires that the value of the attribute be correct, then the programmer would need to update the value herself.

SizeY

The height of the dialog control, in dialog units, when the ooRexx object that represents the control is created.

Note: The `SizeY` attribute is not changed by ooDialog after the object is instantiated. If the programmer resizes the dialog control manually, the programmer would also need to update the value of this attribute herself, if she requires that the value remain correct.

Methods:

Instances of the DialogControl class implement the methods listed in the following DialogControl Instance Methods table.

Table 5-1. DialogControl Instance Methods

Method...	...on page
AbsRect2LogRect	AbsRect2LogRect
AssignFocus	AssignFocus
AssignWindow	AssignWindow
CaptureMouse	CaptureMouse
Clear	Clear
ClearRect	ClearRect
ClientToScreen	ClientToScreen
ConnectFKeyPress	ConnectFKeyPress
ConnectKeyPress	ConnectKeyPress
CreateBrush	CreateBrush
CreateFont	CreateFont
CreatePen	CreatePen
Cursor_AppStarting	Cursor_AppStarting
Cursor_Arrow	Cursor_Arrow
Cursor_Cross	Cursor_Cross
Cursor_No	Cursor_No
CursorPos	CursorPos
Cursor_Wait	Cursor_Wait
DeleteFont	DeleteFont
DeleteObject	DeleteObject
Disable	Disable
DisconnectKeyPress	DisconnectKeyPress
Display	Display
Draw	Draw
DrawAngleArc	DrawAngleArc
DrawArc	DrawArc
DrawLine	DrawLine
DrawPie	DrawPie
DrawPixel	DrawPixel
Enable	Enable

Method...	...on page
FillDrawing	FillDrawing
FontColor	FontColor
FontToDC	FontToDC
ForegroundWindow	ForegroundWindow
FreeDC	FreeDC
GetArcDirection	GetArcDirection
GetClientRect	GetClientRect
GetDC	GetDC
GetFocus	GetFocus
GetID	GetID
GetMouseCapture	GetMouseCapture
GetPixel	GetPixel
GetPos	GetPos
GetRect	GetRect
GetSize	GetSize
GetStyleRaw	GetStyleRaw
GetTextSize	GetTextSize
Group	Group
HasKeyPressConnection	HasKeyPressConnection
Hide	Hide
HideFast	HideFast
HScrollPos	HScrollPos
IsMouseButtonDown	IsMouseButtonDown
LoadBitmap	LoadBitmap
LogRect2AbsRect	LogRect2AbsRect
Move	Move
ObjectToDC	ObjectToDC
OpaqueText	OpaqueText
ProcessMessage	ProcessMessage
Rectangle	Rectangle
Redraw	Redraw
RedrawClient	RedrawClient
RedrawRect	RedrawRect
ReleaseMouseCapture	ReleaseMouseCapture
RemoveBitmap	RemoveBitmap
Resize	Resize
RestoreCursorShape	RestoreCursorShape
ScreenToClient	ScreenToClient

Method...	...on page
Scroll	Scroll
SetArcDirection	SetArcDirection
SetColor	SetColor
SetCursorPos	SetCursorPos
SetFocus	SetFocus
SetFocusToWindow	SetFocusToWindow
SetFont	SetFont
SetForegroundWindow	SetForegroundWindow
SetHScrollPos	SetHScrollPos
SetVScrollPos	SetVScrollPos
SetRect	SetRect
SetTitle	SetTitle
Show	Show
ShowFast	ShowFast
TabStop	TabStop
TabToNext	TabToNext
TabToPrevious	TabToPrevious
Title	Title
Title=	Title=
TransparentText	TransparentText
Update	Update
Value	Value
Value=	Value=
VScrollPos	VScrollPos
Write	Write
WriteToButton	WriteToButton
WriteToWindow	WriteToWindow

5.1. Appearance and Behavior Methods

The following methods deal with the appearance or the behavior of the dialog control. The list contains methods to get or change the size, position, visibility, behavior, etc., of dialog controls.

5.1.1. Show

```
>>-aDialogControl~Show----------------------------------------><
```

The Show method makes a dialog control visible and activates it.

Example:

The following example makes the edit control NAME visible and activates it:

```
MyDialog~GetEditControl("NAME")~Show
```

5.1.2. Hide

```
>>-aDialogControl~Hide----------------------------------------><
```

The Hide method makes a dialog or dialog control invisible and activates another dialog or dialog control.

Example:

```
MyDialog~GetEditControl("NAME")~Hide
```

5.1.3. HideFast

```
>>-aDialogControl~HideFast------------------------------------><
```

The HideFast method marks a dialog or dialog control as invisible but does not redraw it. Send the Update method (see page Update) to the dialog or dialog control to force a redraw.

Example:

```
MyDialog~GetEditControl("NAME")~HideFast
...
MyDialog~Update
```

5.1.4. ShowFast

```
>>-aDialogControl~ShowFast------------------------------------><
```

The ShowFast method marks a dialog or dialog control as visible but does not redraw it. Send the Update method (see page Update) to the dialog or dialog control to force a redraw.

Example:

```
MyDialog~GetEditControl("NAME")~ShowFast
...
MyDialog~Update
```

5.1.5. Display

```
                              +-NORMAL---+------+--+
                              |          +-FAST-+  |
>>-aDialogControl~Display(--"--+-DEFAULT--+------+--+--"--)----><
                              |          +-FAST-+  |
                              +-HIDE-----+------+--+
                              |          +-FAST-+  |
                              +-INACTIVE-----------+
```

The Display method displays (shows, makes visible, or invisible) a dialog control as specified.

Argument:

This argument can be one of the following keywords (case is not significant.) If no argument is given, the default keyword is `NORMAL`:

NORMAL

Shows (makes visible) the dialog control. Same as calling the Show method (see page Show).

DEFAULT

Default is an alias for `NORMAL`. The behavior is exactly the same for the two keywords.

HIDE

Hides (makes invisible) the dialog control. Same as calling the Hide method (see page Hide).

INACTIVE

Displays (shows, makes visible,) the dialog control in its current state. The active dialog remains active. When this method is invoked with the `NORMAL` keyword the dialog the control belongs to will be made the active window. If instead the `INACTIVE` keyword is used, the active window will not be changed. (Obviously, if the dialog that the control belongs to is already the active window, the `INACTIVE` keyword behaves exactly the same as the `NORMAL` keyword.

To each keyword, except the `INACTIVE` keyword, you can add `FAST`, separated by a blank, to suppress the redrawing of the dialog control.

Example:

The following statement makes a tree control in the dialog visible without redrawing it.

```
MyDialog~GetTreeControl(IDC_TREECONTROL_FILES)~Display("NORMAL FAST")
```

This is usually done when there are a number of controls to make visible at one time. The programmer would make them all visible without redrawing. Then use the Update method to repaint everything at once.

5.1.6. Enable

```
>>-aDialogControl~Enable---------------------------------------><
```

The Enable method enables a dialog or dialog control to accept user interaction.

Example:

```
MyDialog~GetEditControl("Name")~Enable
```

5.1.7. Disable

```
>>-aDialogControl~Disable------------------------------------><
```

The Disable method disables a dialog or dialog control.

Example:

```
MyDialog~GetEditControl("Name")~Disable
```

5.1.8. Group

```
>>-aDialogControl~Group(--+-----------+--)--------------------><
                          +-wantStyle-+
```

Add or remove the group style for this control. The group style controls how the user can navigate through the dialog using the keyboard. For most dialogs this does not change while the dialog is executing. However, in some dialogs the programmer may want to change the navigation depending on the options the user selects.

Arguments:

The only argument is

wantStyle

A boolean (.true or .false) to indicate whether the control should have or not have the group style. True (the default) indicates the control should have the group style and false indicates the control should not have the style.

Return value:

Negative values indicate the function failed, non-negative values indicate success.

-4 or less

The value is the negated Operating System Error code. The absolute value of the return can be used to look up the error reason in the Windows documentation.

-3

The optional argument to the method is not a boolean.

-2

There is an (internal) problem with the dialog or the dialog handle.

-1

There is an (internal) problem with the resource ID of the control.

0 or greater

The window style of the dialog control prior to adding or removing the group style.

5.1.9. TabStop

```
>>-aDialogControl~TabStop(--+-----------+--)-------------------><
                            +-wantStyle-+
```

Add or remove the tab stop style for the control. When a control has the tabstop style, the user can set the focus to the control by using the tab key. When a control does not have this style, the tab key will skip over the control. Adding or removing this style during the execution of a dialog allows the programmer to alter how the user navigates through the dialog controls.

Arguments:

The only argument is:

wantStyle

A boolean (.true or .false) to indicate whether the control should have or not have the tabstop style. True (the default) indicates the control should have the tabstop style and false indicates the control should not have the style.

Return value:

Negative values indicate the function failed, non-negative values indicate success.

-4 or less

The value is the negated Operating System Error code. The absolute value of the return can be used to look up the error reason in the Windows documentation.

-3

The second argument to the method is not a boolean.

-2

There is an (internal) problem with the dialog or the dialog handle.

-1

There is an (internal) problem with the resource ID of the control.

0 or greater

The window style of the dialog control prior to adding or removing the tabstop style.

5.1.10. Resize

```
>>-aDialogControl~Resize(--width--,--height--+-----------------------------+--)-><
                                             |       +----------------+    |
                                             |       V                |    |
                                             +-,--"----+-HIDEWINDOW-+-+--"-+
                                                       +-SHOWWINDOW-+
                                                       +-NOREDRAW---+
```

The Resize method resizes a dialog or dialog control.

Arguments:

The arguments are:

width

The new width of the dialog or dialog control, in dialog units.

height

The new height of the dialog or dialog control, in dialog units.

showOptions

One or more of the following keywords, separated by blanks:

HIDEWINDOW

The dialog or dialog control is to be made invisible.

SHOWWINDOW

The dialog or dialog control is to be made visible.

NOREDRAW

The dialog or dialog control is to be repositioned without redrawing it.

Return value:

0

Resizing was successful.

1

Resizing failed.

Example:

The following example resizes the tree view control FILES almost to the size of the window and displays the new size:

```
obj = MyDialog~GetTreeControl("FILES")
if obj = .Nil then return
obj~Resize(MyDialog~SizeX -10, MyDialog~SizeY -20)
parse value obj~GetSize with width height
say "New width of window is" width "and new height is" height
```

5.1.11. Move

```
>>-aDialogControl~Move(--xPos--,--yPos--+----------------------------+--)-><
                                        |      +----------------+    |
                                        |      V                |    |
                                        +-,-"----+-HIDEWINDOW-+-+--"-+
                                                 +-SHOWWINDOW-+
                                                 +-NOREDRAW---+
```

The Move method moves the associated dialog or dialog control to the specified position.

Arguments:

The arguments are:

xPos

The new horizontal position of the dialog or dialog control, in dialog units.

yPos

The new vertical position of the dialog or dialog control, in dialog units.

showOptions

One or more of the following keywords, separated by blanks:

HIDEWINDOW

The dialog or dialog control is to be made invisible.

SHOWWINDOW

The dialog or dialog control is to be made visible.

NOREDRAW

The dialog or dialog control is to be repositioned without redrawing it.

Return value:

0

Moving was successful.

1

Moving failed.

Example:

The following example repositions the tree view control FILES to the upper left corner of the window and displays the new position:

```
obj = MyDialog~GetTreeControl("FILES")
if obj = .Nil then return
obj~Move(1,1)
parse value obj~GetPos with x y
say "New horizontal position of window is" x "and new vertical position is" y
```

5.1.12. SetForegroundWindow

```
>>-aDialogControl~SetForegroundWindow(--hwnd--)---------------------------><
```

The SetForegroundWindow method changes the foreground window to the window specified. The new window must be a top-level window or a dialog window. You can not make a dialog control window the foreground window.

Arguments:

The only argument is:

hwnd

The handle to the window that is to become the foreground window.

Return value:

The handle to the previous foreground window, or 0 if this method failed.

Example:

This example makes another dialog the foreground window and returns the window handle of the current foreground window to the caller, or returns -1 if there was an error:

```
::method switchToReportDlg
  expose reportDlg
  currentHwnd = self~setForeGroundwindow(reportDlg~dlgHandle)
  if currentHwnd == 0 then return -1  -- Report an error
  return currentHwnd
```

5.1.13. ForegroundWindow

```
>>-aDialogControl~ForegroundWindow----------------------------><
```

The ForegroundWindow method returns the handle to the current foreground window.

Return value:

The handle to the foreground window, or 0 if this method failed.

5.1.14. GetID

```
>>-aDialogControl~GetID---------------------------------------><
```

The GetID method retrieves the identification number of the associated dialog or dialog control.

Return value:

The numeric ID.

Example:

The following example displays 1 in most cases:

```
say MyDialog~GetButtonControl("IDOK")~GetID
```

5.1.15. GetStyleRaw

```
>>-aDialogControl~GetStyleRaw----------------------------------><
```

The `GetStyleRaw` method retrieves the Windows style flags of the dialog control. The flags are returned in their numeric format. This allows the programmer to determine the current style of any control. However, the programmer needs to be able to look up the numeric value of the window style flags to make use of this functionality. This can be done using the Windows documentation and the Platform SDK.

Return value:

A negative return indicates an error.

-3 or less

The value of the negated Operating System error code. The absolute value of the return can be used to look up the error reason in the Windows documentation.

-2

There is an (internal) problem with the dialog or the dialog handle.

0 or greater

The numeric value of the window style flags for the control.

Example:

The following example gets the style flags for a tree control and then checks to see if drag and drop is enabled and if label editing is enabled.

The value of the style flags has the potential of being larger than can be accommodated by the default numeric digits. Use `numeric digits 11` if the returned value needs to be manipulated, for instance to use `d2x` on the value.

> **Note:** It is not necessary to convert the `style` value in the example to its hexadecimal string representation before using it as an argument in the `BinaryAnd` function. That is just done in the example for display purposes.

```
  ...

  numeric digits 11
  TVS_DISABLEDRAGDROP = "0x0010"
  TVS_EDITLABELS      = "0x0008"

  treeControl = self~getTreeControl(IDC_TREE)
  style = treeControl~getStyleRaw
  style = "0x" || style~d2x~right(8, '0')

  if BinaryAnd( style, TVS_DISABLEDRAGDROP ) <> 0 then str1 = 'disabled'
  else str1 = 'enabled'

  if BinaryAnd( style, TVS_EDITLABELS ) <> 0 then str2 = 'enabled'
  else str2 = 'disabled'

  say 'Tree-view Control styles'
  say '  Raw style:    ' style
  say '  Drag and drop:' str1
  say '  Edit labels:  ' str2

/* Output coule be: */
```

```
Tree-view Control styles
  Raw style:     0x5000002F
  Drag and drop: enabled
  Edit labels:   enabled
```

5.1.16. GetRect

```
>>-aDialogControl~GetRect------------------------------------><
```

Retrieves the dimensions of the rectangle surrounding the associated dialog or dialog control. The coordinates are relative to the upper left corner of the screen and are specified in screen pixels. The order is: left, top, right, bottom; where 'left' and 'top' are the x and y coordinates of the upper left-hand corner of the rectangle, and 'right' and 'bottom' are the coordinates of the bottom right-hand corner.

Return value:

The coordinates of the dialog or dialog control, separated by blanks.

Example:

The following example calculates the width and height of an entry line:

```
parse value MyDialog~GetEditControl("Name")~GetRect with left top,
right bottom
width = right - left
height = bottom - top
```

5.1.17. SetRect

```
>>-aDialogControl~SetRect(--x--,--y--,--width--,--height-->

>--+-----------------------------+--)-------------------><
   |       +----------------+    |
   |       V                |    |
   +-,--"----+-NOMOVE-----+-+--"-+
             +-NOSIZE-----+
             +-HIDEWINDOW-+
             +-SHOWWINDOW-+
             +-NOREDRAW---+
```

The SetRect method sets new coordinates for the associated dialog or dialog control.

Arguments:

The arguments are:

x,y

The new position of the upper left corner, in screen pixels.

width

The new width of the dialog or dialog control, in screen pixels.

height

The new height of the dialog or dialog control, in screen pixels.

showOptions

One or more of the following keywords, separated by blanks:

NOMOVE

The upper left position of the dialog or dialog control is not changed.

NOSIZE

The size of the dialog or dialog control is not changed.

HIDEWINDOW

The dialog or dialog control is to be made invisible.

SHOWWINDOW

The dialog or dialog control is to be made visible.

NOREDRAW

The dialog or dialog control is to be repositioned without redrawing it.

Return value:

0

Repositioning was successful.

1

Repositioning failed.

5.1.18. GetClientRect

```
>>-aDialogControl~GetClientRect(--+------+--)------------------><
                                  +-hwnd-+
```

The GetClientRect method returns the client rectangle of a dialog or dialog control in screen pixels. The client coordinates specify the upper left and lower right corners of the client area. Because the client coordinates are relative to the upper left corner of the client area of a dialog or dialog control, the coordinates of the upper left corner are (0,0).

Arguments:

The only argument is:

hwnd

The handle to a dialog or dialog control. If this argument is omitted, the dimensions of the associated dialog or dialog control are returned.

Return value:

The client rectangle in the format "*left top right bottom*", separated by blanks.

5.1.19. GetPos

```
>>-aDialogControl~GetPos-------------------------------------><
```

The GetPos method returns the coordinates of the upper left corner of the dialog or dialog control, in dialog units.

Return value:

The horizontal and vertical position, separated by a blank.

Example:

For an example, see Move.

5.1.20. GetSize

```
>>-aDialogControl~GetSize-------------------------------------><
```

The GetSize method returns the width and height of the dialog or dialog control, in dialog units.

Return value:

The width and height, separated by a blank.

Example:

For an example, see Resize.

5.1.21. AssignFocus

```
>>-aDialogControl~AssignFocus----------------------------------><
```

The AssignFocus method sets the input focus to the associated dialog control.

5.1.22. GetFocus

```
>>-aDialogControl~GetFocus------------------------------------><
```

The GetFocus method returns the handle to the dialog control that currently has the input focus.

Return value:

The handle to the dialog control that has the input focus, or 0 if this method failed.

5.1.23. SetFocus

```
>>-aDialogControl~SetFocus(--hwnd--)---------------------------><
```

The SetFocus method sets the input focus to a dialog control and returns the handle of the control that previously had the focus.

Arguments:

The only argument is:

hwnd

The handle to the dialog control that is to receive the input focus.

Return value:

-1

The method failed.

0

The focus was changed, but the control with the previous focus could not be determined.

Other

The handle to the control with the previous focus.

5.1.24. SetFocusToWindow

```
>>-aDialogControl~SetFocusToWindow(--hwnd--)-------------------><
```

The SetFocusToWindow method moves the input focus to another top-level window or dialog. This has the effect of bringing that window to the foreground. This method returns a handle to the dialog *control* that previously had the input focus.

Arguments:

The only argument is:

hwnd

The handle to the top-level window or dialog that is to receive the input focus.

Return value:

-1

The method failed.

0

The focus was changed, but the control with the previous focus could not be determined.

Other

The handle to the control with the previous focus.

5.1.25. SetColor

```
>>-aDialogControl~SetColor(--background--,--foreground--)------><
```

The SetColor method sets the background and foreground colors of the dialog control.

Arguments:

The arguments are:

background

The color number of the background color.

foreground

The color number of the foreground color.

Return value:

0

The color has been assigned.

1

The selected color was already assigned.

Example:

The following example sets the background color of list box FILES to blue:

```
MyDialog~GetListBox("FILES")~SetColor(4, 15)
```

5.1.26. Update

```
>>-aDialogControl~Update--------------------------------------><
```

The Update method makes the contents of the dialog or dialog control invalid and therefore forces it to be updated.

5.1.27. Title

```
>>-aDialogControl~Title---------------------------------------><
```

The Title method retrieves the title of the dialog or dialog control.

Return value:

0

The title was retrieved.

1

Retrieving the title failed.

5.1.28. Title=

```
>>-aDialogControl~Title=--new_title----------------------------><
```

The Title= method sets the title of the dialog or dialog control.

Arguments:

The only argument is:

new_text

A text string that is to be used as the title or text of the dialog or dialog control.

Return value:

0

The title was set.

1

Setting the title failed.

Example:

The following example changes the label of radio button CHOICE2:

```
MyDialog~GetRadioControl("CHOICE2")~Title="&Object Rexx (preferred choice)"
MyDialog~Redraw
```

5.1.29. SetTitle

```
>>-aDialogControl~SetTitle(--new_title--)---------------------><
```

The SetTitle method sets the title of the dialog or dialog control. It is equal to Title=.

5.2. Miscellaneous Dialog Control Methods

The following methods are general purpose dialog control methods.

5.2.1. ProcessMessage

```
>>-aDialogControl~ProcessMessage(--message--,--firstParam--,--secondParam--)-><
```

The ProcessMessage method sends a Windows message to a dialog control.

Arguments:

The arguments are:

message

The number of the message to be sent to the dialog control.

firstParam,secondParam

Additional arguments specific to the message.

Return value:

The return values are message-specific.

Example:

The following example erases the background of the edit control with the resource ID (symbolic ID) of IDC_EDIT1:

```
dlgc = MyDialog~GetEditControl("IDC_EDIT1")
WM_ERASEBACKGROUND = "14"~x2d
hdc = dlgc~GetDc
dlgc~ProcessMessage(WM_ERASEBACKGROUND, hdc, 0)
dlgc~FreeDC(hdc)
```

Note: Use the Windows documentation and Platform SDK to obtain the Window message numbers.

5.2.2. Value

```
>>-aDialogControl~Value----------------------------------------><
```

The Value method retrieves the current value of a dialog control.

Return value:

The current value set in the dialog control.

Note: See GetValue for more information.

5.2.3. Value=

```
>>-aDialogControl~Value=--new_value----------------------------><
```

The Value= method sets a value for a dialog control.

Arguments:

The only argument is:

new_value

The value assigned to the dialog control.

Example:

The following example selects check box RESTART and deselects check box VERIFY:

```
MyDialog~GetCheckControl("RESTART")~Value=1
MyDialog~GetCheckControl("VERIFY")~Value=0
```

Note: See SetValue for more information.

5.2.4. AssignWindow

```
>>-aDialogControl~AssignWindow(--hwnd--)----------------------><
```

The AssignWindow method connects a dialog or dialog control with an existing object of the PlainBaseDialog or DialogControl class. Note that the connected dialog or dialog control might not support all methods provided by the DialogControl class.

Note Well:

The `AssignWindow` method is used internally to connect ooRexx objects (a `DialogControl` object) to the underlying Windows dialog control. It is not recommended that the programmer use this method unless he throughly understands the ooDialog framework. Otherwise, using this method has the potential for causing unpredictable behavior.

Arguments:

The only argument is:

hwnd

The handle to the dialog or dialog control that you want to assign to the DialogControl object.

Return value:

The handle to the dialog or dialog control that has been assigned, or 0 if the connection failed.

Example:

The following example searches the desktop for a dialog with the title "Monitoring Applications", connects it to the object `dlgc` of the DialogControl class, and then hides the dialog:

```
dlgc = .DialogControl~new
...
whnd = FindWindow("Monitoring Application")
if whnd \= 0 then do
   dlgc~AssignWindow(whnd)
   dlgc~Display("HIDE")
end
```

5.3. Connect Event Methods

The following methods connect dialog control events to methods the programmer defines in the parent dialog class.

5.3.1. ConnectKeyPress

```
>>-aDialogControl~ConnectKeyPress(--msgToRaise--,--keys--+-----------+--)---->&lt;
                                                         +-,-filter--+
```

The `ConnectKeyPress` method connects a key down event, when the control has the keyboard focus, with a programmer defined method in the parent dialog of the control. The underlying Windows dialog must exist before this method can be invoked.

> **Note:** This method is only available on systems running Windows XP or later. On earlier versions of Windows, the method will fail and return -4.

The arguments, return values, and behavior of this method are the same as those for the `ConnectKeyPress` method of the `BaseDialog`. However, the method of the `BaseDialog` connects any key down event when the dialog is the active window and this method of the `DialogControl` only connects the key down events when the specific control has the focus. See the ConnectKeyPress method of the `BaseDialog` for a detailed description of the arguments, return values, and behavior.

Arguments:

The arguments are:

msgToRaise

The method to be invoked for the key down event.

keys

The key, or list of keys, whose key down event will invoke the method specified by `msgToRaise`.

filter

An optional filter to be applied to the key down event.

Return value:

The return values are:

0

Success.

1

The underlying mechanism in the Windows API failed.

-1

A problem with one of the arguments.

-2

An (internal) problem with the dialog window.

-3

An (internal) problem with the dialog administration.

-4

This method is not supported by this version of Windows. Windows XP or later is required.

-5

Memory allocation error in the underlying Windows API.

-6

The maximum number of connections has been reached.

-7

The `msgToRaise` method is already connected to a key down event for this dialog control.

Event Method Arguments

The ooDialog method connected to the key press event will receive the following five arguments in the order listed:

keyCode

The numeric code of the key pressed.

shift

True if a shift key was down at the time of the event and false if the shift key was not down.

control

True if a control key was down at the time of the key press, false if it was not.

alt

True if an alt key was down at the time of the key press, false if it was not.

extraInfo

This argument is a string containing keywords. It supplies extra information about the keyboard state at the time of a key press event. The string will contain some combination of these keywords

numOn

Num Lock was on at the time of the key press event.

numOff

Num Lock was off.

capsOn

Caps Lock was on at the time of the key press event.

capsOff

Caps Lock was off.

scrollOn

Scroll Lock was on at the time of the key press event.

scrollOff

Scroll Lock was off.

lShift

The left shift key was down at the time of the key press event.

rShift

The right shift key was down.

lControl

The left control key was down at the time of the key press event.

rControl

The right control key was down.

lAlt

The left alt key was down at the time of the key press event.

rAlt

The right alt key was down.

Example:

The following example could be from an application that supplies extended editing abilities for a multi-line edit control. When the edit control has the focus the user can use Alt-D to delete the current word, Ctrl-D to delete the current line, and Shift-Alt-D delete the current paragraph:

```
::method initDialog
  expose editControl

  editControl = self~getEditControl(IDC_EDTIT)
  ...

  -- Capture the D key press when either the Alt or Control keys are
  -- also pressed.
  editControl~connectKeyPress(onDPress, self~vCode('D'), "ALT CONTROL")
  ...

::method onDPress
  expose editControl
  use arg key, shift, control, alt, info

  -- Determine which of the key press combinations this is and take
  -- appropriate action if it is a combination we are interested in.

  isAltD = \ shift & \ control & alt
  isCtrlD = \ shift & control & \ alt
  isShiftAltD = shift & \ control & alt

  if isAltD then self~deleteWord(editControl)
  if isCtrlD the self~deleteLine(editControl)
  if isShiftAltD then self~deleteParagraph(editControl)
```

5.3.2. ConnectFKeyPress

```
>>-aDialogControl~ConnectFKeyPress(--msgToRaise--)------------><
```

The `ConnectFKeyPress` method is a convenience method that connects all F key down events, when the control has the keyboard focus, with a programmer defined method in the parent dialog of the control. (This is for the standard function keys, F2 through F24.) The underlying Windows dialog must exist before this method can be invoked.

Note: This method is only available on systems running Windows XP or later. On earlier versions of Windows, the method will fail and return -4.

The arguments, return values, and behavior of this method are the same as those for the `ConnectFKeyPress` method of the `BaseDialog`. However, the method of the `BaseDialog` class connects any F key down event when the dialog is the active window and this method of the `DialogControl` class only connects the F key down events when the specific control has the focus. See the ConnectFKeyPress method of the `BaseDialog` for a more detailed description of the arguments, return values, and behavior.

Arguments:

The required argument is:

msgToRaise

The method to be invoked for the key down event.

Return value:

The return values are:

0

Success.

1

The underlying mechanism in the Windows API failed.

-1

A problem with one of the arguments.

-2

An (internal) problem with the dialog window.

-3

An (internal) problem with the dialog administration.

-4

This method is not supported by this version of Windows. Windows XP or later is required.

-5

Memory allocation error in the underlying Windows API.

-6

The maximum number of connections has been reached.

-7

The `msgToRaise` method is already connected to a key down event for this dialog control.

Event Method Arguments

The ooDialog method connected to the F key down event receives five arguments. This arguments are described in detail in the ConnectKeyPress method of the `DialogControl`.

Example:

The following example connects the F key down events for an edit control. The application determines which F key was pressed and takes appropriate action. For this application, the F3 key is the search key.

```
::method initDialog
  expose editControl searchToken

  editControl = self~getEditControl(IDC_EDIT)
  editControl~connectFKeyPress(onFKey)
  ...
  lastSearchToken = ""
  ...

::method onFKey
  expose editControl searchToken
  use arg key

  select
    when self~keyName(key) == 'F2' then do
      ...
    end
    when self~keyName(key) == 'F3' then do
      description = "What word would you like to search for?"
      searchToken = inputBox(description, "Search", searchToken)

      if searchToken <> "" then
        self~findAndHighlight(searchToken, editControl)
    end
    when self~keyName(key) == 'F4' then do
      ...
    end
    ...
  end
  ...
```

5.3.3. DisconnectKeyPress

```
>>-aDialogControl~DisconnectKeyPress(--+--------------+--)----->&lt;
                                       +--methodName--+
```

The `DisconnectKeyPress` method disconnects a previously connected method with a key down event. If no method name is specified than all key down event connections are removed. Otherwise, only the key down events connected to the specific method are removed.

Note: This method is only available on systems running Windows XP or later. On earlier versions of Windows, the method will fail and return -4.

The arguments, return values, and behavior of this method are the same as those for the `DisonnectKeyPress` method of the `BaseDialog`. However, this method only works for connections made using the `DialogControl's` ConnectKeyPress method. It can not work with a connection made though the `BaseDialog's` method. See the DisconnectKeyPress method of the `BaseDialog` for a more detailed description of the behavior of this method.

Arguments:

The single optional argument is:

methodName

If `methodName` is specified, only the key down events connected to that method are disconnected. If the argument is omitted, then all key down events for the dialog control will be disconnected.

Return value:

The return values are:

0

Success.

1

The underlying mechanism in the Windows API that captures the key down events failed to disconnect.

-1

The specified `methodName` is not connected to any key down events for this control.

-2

The underlying mechanism in the Windows API that captures the key down events was never installed.

-4

This method is not supported by this version of Windows. Windows XP or later is required.

Example:

The following example could come from a dialog where the user can enable or disable the user of hot keys when she is working within an edit control. When the user presses the disable hot keys button, the dialog disables the hot keys for the edit control by removing the key press connections.

```
::method defineDialog
```

```
    ...
    self~addButton(IDC_PB_DISABLE, 60, 135, 65, 15, "Disable Hot Keys", onDisable)
    ...

  method onDisable
    editControl = self~getEditControl(IDC_EDIT)
    editControl~disconnectKeyPress
```

5.3.4. HasKeyPressConnection

```
>>-aDialogControl~HasKeyPressConnection(--+--------------+--)----->&lt;
                                          +--methodName--+
```

This method is used to query if a connection to a key down event already exists.

> **Note:** This method is only available on systems running Windows XP or later. On earlier versions of Windows, the method will fail and return -4.

The arguments, return values, and behavior of this method are the same as those for the `HasKeyPressConnection` method of the `BaseDialog`. However, this method only works for connections made using the `DialogControl's` ConnectKeyPress method. It can not work with a connection made though the `BaseDialog's` method. The HasKeyPressConnection method of the `BaseDialog` may provide some additional insight into the behavior of this method.

Arguments:

The single optional argument is:

methodName

When this argument is not used, `HasKeyPressConnection` queries if any key down event for the dialog control is connected to a method. When `methodName` is specified, the query checks for a connection to the specified method.

Return value:

The returned value is always `true` or `false`.

true

A connection exists.

false

No connection exists.

Example:

The following example is from an application where the user can enable the use of hot keys when an edit control has the focus, or not. The reset push button is used in the application to reset the state of the dialog. One of the things done when the state is reset to check or uncheck a check box that shows whether hot keys are currently enabled or not.

```
::method defineDialog

  ...
  self~addCheckBox(IDC_CHECK_FKEYSENABlED, , 30, 60, , , "Hot Keys Enabled")
  ...
  self~addButton(IDC_PB_RESET, 60, 135, 45, 15, "Reset", onReset)
  ...

::method onReset

  ...
  editControl = self~getEditControl(IDC_EDIT)

  if editControl~hasKeyPressConnection then
    self~getCheckControl(IDC_CHECK_FKEYSENABlED)~check
  else
    self~getCheckControl(IDC_CHECK_FKEYSENABlED)~uncheck
  ...
```

5.4. Draw Methods

The following methods are used to draw, redraw, and clear a dialog or dialog control.

5.4.1. Draw

```
>>-aDialogControl~Draw----------------------------------------><
```

The Draw method draws the dialog or dialog control.

Return value:

0

Drawing was successful.

1

Drawing failed.

5.4.2. Clear

```
>>-aDialogControl~Clear---------------------------------------><
```

The Clear method draws the dialog or dialog control using the background brush.

Return value:

0

Clearing was successful.

1

Clearing failed.

5.4.3. ClearRect

```
>>-aDialogControl~ClearRect(--left--,--top--,--right--,--bottom--)-><
```

The ClearRect method draws the dialog or dialog control using the background brush.

Arguments:

The arguments are:

left,top

The upper left corner of the rectangle, in screen pixels.

right,bottom

The lower right corner of the rectangle, in screen pixels.

Return value:

0

Drawing was successful.

1

Drawing failed.

5.4.4. Redraw

```
>>-aDialogControl~Redraw--------------------------------------><
```

The Redraw method redraws the dialog or dialog control immediately.

Return value:

0

Redrawing was successful.

1

Redrawing failed.

5.4.5. RedrawRect

```
>>-aDialogControl~RedrawRect(--left--,--top--,--right--,--bottom-->

>--+-----------+--)--------------------------------------------><
   +-,erasebkg-+
```

The RedrawRect method immediately redraws the rectangle of the client area of the associated dialog. You can specify whether the background of the dialog is to be erased before redrawing.

Arguments:

The arguments are:

left,top

The upper left corner of the rectangle relative to the client area, in screen pixels.

right,bottom

The lower right corner of the rectangle relative to the client area, in screen pixels.

erasebkg

If this argument is 1 or "Y", the background of the dialog is erased before repainting.

Return value:

0

Redrawing was successful.

1

Redrawing failed.

5.4.6. RedrawClient

```
>>-aDialogControl~RedrawClient(--erasebkg--)------------------><
```

The RedrawClient method immediately redraws the entire client area of the dialog or dialog control. You can specify whether the background of the dialog or dialog control is to be erased before redrawing.

Arguments:

The only argument is:

erasebkg

If you specify 1 or "Y", the background of the dialog is erased before redrawing.

Return value:

0

Redrawing was successful.

1

Redrawing failed.

5.5. Conversion Methods

The following methods are used to convert and map coordinates of a dialog or dialog control.

5.5.1. LogRect2AbsRect

```
>>-aDialogControl~LogRect2AbsRect(--left--,--top--,--right--,--bottom--)-><
```

The LogRect2AbsRect method converts the coordinates from dialog units to screen pixels.

Arguments:

The arguments are:

left,top

The position of the upper left corner, in dialog units.

right,bottom

The position of the lower right corner, in dialog units.

Return value:

A compound variable that stores the four screen pixel coordinates. The position of the upper left corner is stored in RetStem.left and RetStem.top. The position of the lower right corner is stored in RetStem.right and RetStem.bottom. The tails left, top, right, and bottom must be uninitialized symbols.

Example:

```
absrect. = MyDialog~LogRect2AbsRect(5, 5, 10, 10)
say "Screen pixel rectangle=" absrect.left "," absrect.top ",",
absrect.right "," absrect.bottom
```

5.5.2. AbsRect2LogRect

```
>>-aDialogControl~AbsRect2LogRect(--left--,--top--,--right--,--bottom--)-><
```

The AbsRect2LogRect method converts the coordinates from screen pixels to dialog units.

Arguments:

The arguments are:

left,top

The position of the upper left corner, in screen pixels.

right,bottom

The position of the lower right corner, in screen pixels.

Return value:

A compound variable that stores the four screen dialog units. The position of the upper left corner is stored in RetStem.left and RetStem.top. The position of the lower right corner is stored in RetStem.right and RetStem.bottom. The tails left, top, right, and bottom must be uninitialized symbols.

Example:

```
rectdunit. = MyDialog~AbsRect2LogRect(20, 20, 40, 40)
say "Dialog unit rectangle=" rectdunit.left "," rectdunit.top ",",
rectdunit.right "," rectdunit.bottom
```

5.5.3. ScreenToClient

```
>>-aDialogControl~ScreenToClient(--x--,y--)-------------------><
```

The ScreenToClient method maps the coordinates relative to the upper left corner of the screen, to a location within the client area relative to the upper left corner of the dialog's client area.

Arguments:

The arguments are:

x

The horizontal position, in screen pixels.

y

The vertical position, in screen pixels.

Return value:

The horizontal and vertical positions of the specified location relative to the upper left corner of the client area, separated by a blank.

5.5.4. ClientToScreen

```
>>-aDialogControl~ClientToScreen(--x--,y--)-------------------><
```

The ClientToScreen method maps the coordinates relative to the dialog's client area to the coordinates relative to the upper left corner of the screen.

Arguments:

The arguments are:

x

The horizontal position in screen pixels.

y

The vertical position in screen pixels.

Return value:

The horizontal and vertical positions of the specified location relative to the location (0,0) of the screen, separated by a blank.

5.6. Scroll Methods

The following methods are used to scroll a dialog or dialog control and to set scroll bars.

5.6.1. Scroll

```
>>-aDialogControl~Scroll(--cx--,--cy--)------------------------><
```

The Scroll method scrolls the contents of the associated dialog or dialog control by the amount specified.

Arguments:

The arguments are:

cx

The number of screen pixels the content of the dialog or dialog control is to be scrolled to the right or to the left, if negative.

cy

The number of screen pixels the content of the dialog or dialog control is to be scrolled downward or upward, if negative.

Return value:

0

Scrolling was successful.

1

Scrolling failed.

5.6.2. HScrollPos

```
>>-aDialogControl~HScrollPos----------------------------------><
```

The HScrollPos method returns the position of the horizontal scroll bar in the associated dialog or dialog control.

Return value:

The position of the horizontal scroll bar.

5.6.3. VScrollPos

```
>>-aDialogControl~VScrollPos----------------------------------><
```

The VScrollPos method returns the position of the vertical scroll bar in the associated dialog or dialog control.

Return value:

The position of the vertical scroll bar.

5.6.4. SetHScrollPos

```
                                                +-1------+
>>-aDialogControl~SetHScrollPos(--position--,--+--------+--)---><
                                                +-redraw-+
```

The SetHScrollPos method sets the thumb position of the horizontal scroll bar contained in the associated dialog or dialog control.

Arguments:

The arguments are:

position

The new thumb position of the horizontal scroll bar.

redraw

If this argument is 1 (the default), the display of the scroll bar is updated.

Return value:

The previous position of the horizontal scroll bar, or 0 if this method failed.

5.6.5. SetVScrollPos

```
                                              +-1------+
>>-aDialogControl~SetVScrollPos(--position--,--+--------+--)---><
                                              +-redraw-+
```

The SetVScrollPos method sets the thumb position of the vertical scroll bar contained in the associated dialog or dialog control.

Arguments:

The arguments are:

position

The new thumb position of the vertical scroll bar.

redraw

If this argument is 1 (the default), the display of the scroll bar is updated.

Return value:

The previous position of the vertical scroll bar, or 0 if this method failed.

5.7. Mouse and Cursor Methods

The following methods are used to position and shape the mouse cursor and to capture the mouse.

5.7.1. CursorPos

```
>>-aDialogControl~CursorPos----------------------------------><
```

The CursorPos method returns the current position of the mouse cursor.

Return value:

The horizontal and vertical position of the mouse, separated by a blank.

Example:

See SetCursorPos for an example on how to use this method.

5.7.2. SetCursorPos

```
>>-aDialogControl~SetCursorPos(--x--,y--)---------------------><
```

The SetCursorPos method moves the mouse cursor to the specified position. This method can be used to force the repainting of the mouse cursor or to keep the mouse cursor within a specific rectangle.

Arguments:

The arguments are:

x

The horizontal position of the mouse cursor, in screen pixels.

y

The vertical position of the mouse cursor, in screen pixels.

Return value:

0

Moving the mouse cursor was successful.

1

Moving the mouse cursor failed.

Example:

The following example shows two methods: one indicating that processing has started and one indicating that processing has completed. The method IndicateBeginProcessing changes the shape of the mouse cursor to the WAIT cursor and IndicateEndProcessing restores the original mouse cursor shape. Both methods retrieve the current position of the mouse cursor and move it by one screen pixel in each direction to force the repainting of the mouse cursor.

```
::method IndicateBeginProcessing
  self~Current_Cursor = self~Cursor_Wait
  parse value self~CursorPos with curx cury
  self~SetCursorPos(curx+1, cury+1)

::method IndicateEndProcessing
  self~RestoreCursorShape(self~Current_Cursor)
  parse value self~CursorPos with curx cury
  self~SetCursorPos(curx-1, cury-1)
```

See DefListDragHandler for another example on how to use the mouse methods.

5.7.3. RestoreCursorShape

```
>>-aDialogControl~RestoreCursorShape(--+--------------+--)----->&lt;
                                       +-CursorHandle-+
```

The RestoreCursorShape method restores the shape of the mouse cursor.

Arguments:

The only argument is:

CursorHandle

The handle to the mouse cursor shape returned by the Cursor_Arrow, Cursor_AppStarting, Cursor_Cross, Cursor_No, or Cursor_Wait method.

If you omit this argument, the cursor shape is set to an arrow. Therefore, it is recommended that you store the original mouse cursor shape by specifying its handle when you change its shape.

Return value:

The handle to the current cursor shape, that is, the shape that was used before the cursor was restored to the given shape.

Example:

See SetCursorPos for an example on how to use this method.

5.7.4. Cursor_Arrow

```
>>-aDialogControl~Cursor_Arrow--------------------------------><
```

The Cursor_Arrow method sets the shape of the mouse cursor to the standard arrow. The new shape is only used when the mouse cursor is within the rectangle of the associated dialog or dialog control.

Return value:

The handle to the current cursor shape, that is, the shape that was used before the arrow shape was set.

Example:

See SetCursorPos for an example on how to use this method.

5.7.5. Cursor_AppStarting

```
>>-aDialogControl~Cursor_AppStarting---------------------------><
```

The Cursor_AppStarting method sets the shape of the mouse cursor to the standard arrow with a small hourglass. The new shape is only used when the mouse cursor is within the rectangle of the associated dialog or dialog control.

Return value:

The handle to the current cursor shape, that is, the shape that was used before the arrow shape with the hourglass was set.

Example:

See SetCursorPos for an example on how to use this method.

5.7.6. Cursor_Cross

```
>>-aDialogControl~Cursor_Cross--------------------------------><
```

The Cursor_Cross method sets the shape of the mouse cursor to a crosshair. The new shape is only used when the mouse cursor is within the rectangle of the associated dialog or dialog control.

Return value:

The handle to the current shape, that is, the shape that was used before the crosshair cursor shape was set.

Example:

See SetCursorPos for an example on how to use this method.

5.7.7. Cursor_No

```
>>-aDialogControl~Cursor_No-----------------------------------><
```

The Cursor_No method sets the shape of the mouse cursor to a slashed circle to deny access. The new shape is only used when the mouse cursor is within the rectangle of the associated dialog or dialog control.

Return value:

The handle to the current shape, that is, the shape that was used before the slashed-circle cursor shape was set.

Example:

See SetCursorPos for an example on how to use this method.

5.7.8. Cursor_Wait

```
>>-aDialogControl~Cursor_Wait--------------------------------><
```

The Cursor_Wait method sets the shape of the mouse cursor to the hourglass. The new shape is only used when the mouse cursor is within the rectangle of the associated dialog or dialog control.

Return value:

The handle to the current shape, that is, the shape that was used before the hourglass shape was set.

Example:

See SetCursorPos for an example on how to use this method.

5.7.9. GetMouseCapture

```
>>-aDialogControl~GetMouseCapture-----------------------------><
```

The GetMouseCapture method retrieves the dialog or dialog control that captured the mouse. This dialog or dialog control receives the entire mouse input regardless of whether the mouse cursor is within the borders of the dialog or dialog control.

Return value:

The handle to the dialog or dialog control that captures the mouse, or 0 if the mouse is not captured.

5.7.10. CaptureMouse

```
>>-aDialogControl~CaptureMouse--------------------------------><
```

The CaptureMouse method captures the mouse. This means that the associated dialog or dialog control receives the entire mouse input regardless of whether the mouse cursor is within the borders of the dialog or dialog control.

Return value:

The handle to the dialog or dialog control that previously captured the mouse, or 0 if the mouse was not captured before.

Note: If you change the cursor shape while the mouse is being captured, this change is ignored.

5.7.11. ReleaseMouseCapture

```
>>-aDialogControl~ReleaseMouseCapture-------------------------><
```

The ReleaseMouseCapture method releases the mouse capture from a dialog or dialog control and restores normal mouse input processing. This means that the mouse input is then received by another dialog or dialog control that captured the mouse.

Return value:

0

The mouse capture was released.

1

Releasing the mouse capture failed.

5.7.12. IsMouseButtonDown

```
>>-aDialogControl~IsMouseButtonDown(--"--+-LEFT---+--"--)------><
                                        +-MIDDLE-+
                                        +-RIGHT--+
```

The IsMouseButtonDown method retrieves information on whether a mouse button is pressed.

Arguments:

The only argument is:

button

The location of the mouse button you are interested in.

Return value:

0

The button is not being pressed.

1

The button is being pressed.

5.8. Bitmap Methods

The following methods load and release bitmaps.

5.8.1. LoadBitmap

```
>>-aDialogControl~LoadBitmap(--bmpFilename--+-------------+--)-><
                                            +-,--"USEPAL"-+
```

The LoadBitmap method loads a bitmap from a file into memory and returns the handle to the bitmap.

Arguments:

The arguments are:

bmpFilename

The name of a bitmap file. The name can also include a relative or absolute path.

USEPAL

Sets the color palette of the bitmap as the system color palette.

Example:

The following example loads into memory the bitmap file, `Walker.bmp`, which is located in the `BMP` subdirectory. hBmp is the handle to this in-memory bitmap.

```
hBmp = MyDialog~LoadBitmap("bmp\Walker.bmp", "USEPAL")
```

Note: Do not forget to call the RemoveBitmap method to free memory when the bitmap is no longer in use. You have to specify the INMEMORY option when using the ConnectBitmapButton or ChangeBitmapButton method.

5.8.2. RemoveBitmap

```
>>-aDialogControl~RemoveBitmap(--hBitmap--)-------------------><
```

The RemoveBitmap method frees an in-memory bitmap that was loaded by LoadBitmap.

Arguments:

The only argument is:

hBitmap

The bitmap handle.

5.9. Device Context Methods

The following methods are used to retrieve and release a device context (DC).

A device context (DC) is associated with a dialog or dialog control, and is a drawing area managed by a dialog or dialog control. It stores information about graphic objects (such as bitmaps, lines, and pixels that are displayed) and the tools (such as pens, brushes, and fonts) that are used to display them.

5.9.1. GetDC

```
>>-aDialogControl~GetDC---------------------------------------><
```

The GetDC method reserves drawing resources and returns the handle to the display device context of a dialog or dialog control.

Return value:

The handle to the device context, or 0 if this method failed.

Example:

The following example retrieves the device context of button DRAWINGS, processes the drawing commands, and frees the device context resources:

```
obj = MyDialog~GetButtonControl("DRAWINGS")
if obj = .Nil then return -1
dc = obj~GetDC
if dc = 0 then return -1
... /* draw something */
obj~FreeDC(dc)
```

Note: When you have finished with the device context, call FreeDC.

5.9.2. FreeDC

```
>>-aDialogControl~FreeDC(--dc--)------------------------------><
```

The FreeDC method releases the device context resources that were reserved for GetDC.

Arguments:

The only argument is:

dc

The handle to the device context that is to be released.

Return value:

0

The device context resources were released.

1

Releasing the device context resources failed.

Example:

See GetDC for an example.

Note: Always call this method when you have finished with the device context.

5.10. Text Methods

The following methods are used to display text dynamically in a window area and to modify the state of a device context. See GetWindowDC, GetDC, and GetButtonDC for information on how to retrieve a device context.

5.10.1. Write

```
>>-aDialogControl~Write(--xPos--,--yPos--,--text---------------->

>--+-------------------------------------+-------------------->
   |    +-"SYSTEM"-+                      |
   +-,--+----------+--+-----------------+-+
        +-fontName-+  |    +-10-------+ |
                      +-,--+----------+-+
                           +-fontSize-+

>--+-------------------------------------------------------------------------+--)-><
   +-+-----------------------------------------------------------------------+-+
     +-+---------------------------------------------------------------------+-+
       |       +-----------------+                                           |
       |       V +-OPAQUE------+ |                                           |
       +-,--"----+-THIN--------+-+--"--+--------------------------------+-+
                 +-EXTRALIGHT--+       +-,--+---------+--+------------+-+
                 +-LIGHT-------+            +-fgColor-+  +-,--bkColor-+
                 +-MEDIUM------+
                 +-SEMIBOLD----+
                 +-EXTRABOLD---+
                 +-BOLD--------+
```

```
+-HEAVY-------+
+-UNDERLINE---+
+-ITALIC------+
+-STRIKEOUT---+
+-TRANSPARENT-+
+-CLIENT------+
```

The Write method writes the specified text to the device context associated with the dialog or dialog control in the given font, style, and color at the given position.

Arguments:

The arguments are:

xPos, yPos

The starting position of the text, in pixels.

text

The string that you want to write to the dialog or dialog control.

fontName

The name of a font. If omitted, the SYSTEM font is used.

fontSize

The size of the font. If omitted, the standard size (10) is used.

fontStyle

One or more of the keywords listed in the syntax diagram, separated by blanks:

TRANSPARENT

This style writes the text without clearing the background.

OPAQUE

This style, which is the default, clears the background with the given background color, or with white if the background color is omitted, before writing the text.

CLIENT

The device context of the client area of the dialog or dialog control is used instead of the device context of the entire dialog or dialog control.

fgcolor

The color index of the text color.

bkColor

The color index of the background color. The background color is not used in transparent mode.

Example:

The following example writes the string "Hello world!" to the dialog using a blue 24pt Arial font in bold and transparent, italic style:

```
MyDialog~Write(5, 5, "Hello world!", "Arial", 24, ,
"BOLD ITALIC TRANSPARENT CLIENT", 4)
```

5.10.2. WriteDirect

```
>>-aDialogControl~WriteDirect(--dc--,--xPos--,--yPos--,--text--)-><
```

The WriteDirect method enables you to write text to a device context at a given position.

Arguments:

The arguments are:

dc

A device context.

xPos, yPos

The position where the text is placed, in pixels.

text

The string you want to write to the dialog or dialog control.

5.10.3. TransparentText

```
>>-aDialogControl~TransparentText(--dc--)---------------------->>
```

The TransparentText method enables you to write text to a device context using WriteDirect in transparent mode, that is, without a white background behind the text. Restore the default mode using OpaqueText.

Arguments:

The only argument is:

dc

A device context.

5.10.4. OpaqueText

```
>>-aDialogControl~OpaqueText(--dc--)--------------------------><
```

The OpaqueText method restores the default text mode, that is, with a white background behind the text, which overlays whatever is at that position in the dialog or dialog control. Use this method after transparent mode was set using TransparentText.

Arguments:

The only argument is:

dc

A device context.

5.10.5. WriteToWindow

```
>>-aDialogControl~WriteToWindow(--hwnd--,--xPos--,--yPos--,--text-->

>--+--------------------------------------+--------------------->
   |    +-"SYSTEM"-+                      |
   +-,--+----------+--+-----------------+-+
        +-fontName-+  |    +-10-------+ |
                      +-,--+----------+-+
                           +-fontSize-+

>--+-----------------------------------------------------------------------+--)-><
   +-+-------------------------------------------------------------------+-+
     +-+---------------------------------------------------------------+-+
       |       +-----------------+                                     |
       |       V +-OPAQUE------+ |                                     |
       +-,--"----+-THIN--------+-+--"--+-------------------------------+-+
                 +-EXTRALIGHT--+       +-,--+---------+--+------------+-+
                 +-LIGHT-------+            +-fgColor-+  +-,--bkColor-+
                 +-MEDIUM------+
                 +-SEMIBOLD----+
                 +-EXTRABOLD---+
                 +-BOLD--------+
                 +-HEAVY-------+
```

```
+-UNDERLINE---+
+-ITALIC------+
+-STRIKEOUT---+
+-TRANSPARENT-+
+-CLIENT------+
```

The WriteToWindow method enables you to write text to a dialog or dialog control in the given font and size to the given position.

Arguments:

The arguments are:

hwnd

The handle of the dialog or dialog control. See Get for how to get a valid handle.

xPos, yPos

The starting position of the text, in pixels.

text

The string you want to write to the dialog or dialog control.

fontName

The name of a font. If omitted, the SYSTEM font is used.

fontSize

The size of the font. If omitted, the standard size (10) is used.

fontStyle

One or more of the keywords listed in the syntax diagram, separated by blanks:

TRANSPARENT

This style writes the text without clearing the background.

OPAQUE

This style, which is the default, clears the background with the given background color, or with white if the background color is omitted, before writing the text.

CLIENT

The device context of the dialog's client area is used instead of the device context of the entire dialog.

Example:

This example writes the string "Hello world!" to the dialog window using a 24pt Arial font in bold and italic style:

```
hwnd=MyDialog~Get
MyDialog~WriteToWindow(hwnd, 23, 15, "Hello world!", ,
                       "Arial", 24, "BOLD ITALIC")
```

5.10.6. WriteToButton

```
>>-aDialogControl~WriteToButton(--id--,--xPos--,--yPos--,--text-->

>--+-------------------------------------+--------------------->
   |    +-"SYSTEM"-+                     |
   +-,--+----------+--+-----------------+-+
        +-fontName-+  |    +-10-------+ |
                      +-,--+----------+-+
                           +-fontSize-+

>--+-----------------------------------------------------------------------+--)-><
   +-+-------------------------------------------------------------------+-+
     +-+---------------------------------------------------------------+-+
       |       +-----------------+                                     |
       |       V +-OPAQUE------+ |                                     |
       +-,--"----+-THIN--------+-+--"--+-------------------------------+-+
                 +-EXTRALIGHT--+       +-,--+---------+--+------------+-+
                 +-LIGHT-------+            +-fgColor-+  +-,--bkColor-+
                 +-MEDIUM------+
                 +-SEMIBOLD----+
                 +-EXTRABOLD---+
                 +-BOLD--------+
                 +-HEAVY-------+
                 +-UNDERLINE---+
                 +-ITALIC------+
                 +-STRIKEOUT---+
                 +-TRANSPARENT-+
                 +-CLIENT------+
```

The WriteToButton method enables you to write text to a button in the given font and size to the given position.

Arguments:

The arguments are:

id

The ID of a button.

See WriteToWindow for a description of the other arguments.

5.10.7. GetTextSize

```
>>-aDialogControl~GetTextSize(--text--+-------------------------------------->
                                      +--,--fontname--+--------------------->
                                                      +--,--fontsize--+----->

>--------------------+--)------------------------------------------------><
>-----------------+--+
>--+-----------+--+
   +--,--hwnd--+
```

The GetTextSize method returns the width and height that the specified text requires in the font and size given.

Arguments:

The arguments are:

text

The text for which the dimensions are to be returned.

fontname

The name of the font used in the device context (DC).

fontsize

The size of the font used in the device context (DC).

hwnd

The handle to the dialog or dialog control that is the owner of the device context (DC).

Return value:

The width and height of the text, in dialog units, separated by a blank.

Example:

The following example stores the space required by the specified text in the device context of MyButton, in cx and cy:

```
parse value MyButton~GetTextSize("This is the output!") with cx cy
```

5.10.8. SetFont

```
<>>-aDialogControl~SetFont(--fontHandle--+---------+--)--------><
                                         +-,redraw-+
```

The SetFont method assigns another font to the text in a dialog or dialog control.

Arguments:

The arguments are:

fontHandle

The handle to the font that is to be used by the dialog or dialog item. Use CreateFont to get this handle.

redraw

If you specify 1, the dialog or dialog item is redrawn.

Example:

The following example creates the font Arial with a pitch size of 14 and assigns it to the tree view control FILES, which is forced to be redrawn.

```
hfnt = Mydialog~CreateFont("Arial", 14)
MyDialog~GetTreeControl("FILES")~SetFont(hfnt, 1)
```

5.10.9. CreateFont

```
<                              +-"SYSTEM"-+     +-10-------+
>>-aDialogControl~CreateFont(--+----------+--,--+----------+--,-->
                               +-fontName-+     +-fontSize-+

      +----------------+
      V                |
>--"----+-THIN-------+-+--"--+--------------+--)---------------><
        +-EXTRALIGHT-+       +-,--fontWidth-+
        +-LIGHT------+
        +-MEDIUM-----+
        +-SEMIBOLD---+
        +-EXTRABOLD--+
        +-BOLD-------+
        +-HEAVY------+
        +-UNDERLINE--+
        +-ITALIC-----+
        +-STRIKEOUT--+
```

The CreateFont method creates a font. It returns a handle that you can use in the FontToDC method (see page FontToDC) to activate the font in a device context or in the SetItemFont method (see page SetItemFont) to change the font of a dialog or dialog item.

Arguments:

The arguments are:

fontName

The name of a font. You can look for valid fonts in the Fonts folder of your Windows Control Panel. If omitted, the SYSTEM font is used.

fontSize

The size of the font. If omitted, the standard size (10) is used.

fontStyle

One or more of the keywords listed in the syntax diagram, separated by blanks.

fontWidth

This argument is optional if it differs from fontSize.

Example:

The following example creates a 16-point italic Arial font:

```
hfnt = MyDialog~CreateFont("Arial", 16, "ITALIC")
```

5.10.10. DeleteFont

```
>>-aDialogControl~DeleteFont(--hFont--)------------------------><
```

The DeleteFont method deletes a font. This method is to be used to delete a font created with the CreateFont method (see page CreateFont).

Arguments:

The only argument is:

hFont

The handle of a font.

5.10.11. FontToDC

```
>>-aDialogControl~FontToDC(--dc--,hFont--)---------------------><
```

The FontToDC method loads a font into a device context and returns the handle of the previous font. Use the GetWindowDC, GetDC, or GetButtonDC method to retrieve a device context, and the CreateFont method to get a font handle. To reset the font to the original state, use another FontToDC call with the handle of the previous font. To release the device context, use the FreeWindowDC, FreeDC, or FreeButtonDC method.

Arguments:

The arguments are:

dc

The device context of a dialog or button.

hFont

The handle of a font.

Example:

This example loads an Arial font into the current dialog window:

```
hfnt = MyDialog~CreateFont("Arial", 16, "ITALIC")
dc   = MyDialog~GetDC
oldf = MyDialog~FontToDC(dc,hfnt)  /* activate font */
...
MyDialog~FontToDC(dc,oldf)         /* restore previous font */
MyDialog~FreeDC(dc)
```

5.10.12. FontColor

```
>>-aDialogControl~FontColor(--color--,dc--)-------------------><
```

The FontColor method sets the font color for a device context.

Arguments:

The arguments are:

color

The index of a color in the system's color palette.

dc

The device context.

5.11. Graphic Methods

These methods deal with drawing graphics within the device context of a window. See GetWindowDC, GetDC, and GetButtonDC for information on how to retrieve a device context.

5.11.1. CreateBrush

```
>>-aDialogControl~CreateBrush(--color--+-----------------+--)-->
                                       +-,brushSpecifier-+
```

The CreateBrush method creates a color brush or a bitmap brush. It returns the handle to a brush object. To remove the brush, use DeleteObject. The brush is used to fill rectangles.

Arguments:

The arguments are:

color

The color number. For a list of color numbers, refer to Definition of Terms.

brushSpecifier

The name of a bitmap file or one of the following keywords to create a hatched brush:

UPDIAGONAL

A 45-degree upward, left-to-right hatch

CROSS

A horizontal and vertical crosshatch

DIAGCROSS

A 45-degree crosshatch

DOWNDIAGONAL

A 45-degree downward, left-to-right hatch

HORIZONTAL

A horizontal hatch

VERTICAL

A vertical hatch

If CreateBrush is sent to a dialog object (subclass of ResDialog), *brushSpecifier* can also be an integer resource ID for a bitmap stored in the DLL that also stores the resource.

If this argument is omitted, a solid brush with the specified color is created.

5.11.2. CreatePen

```
                              +-1-----+
>>-aDialogControl~CreatePen(--+-------+--,--+----------------------+--,-->
                              +-width-+     |    +-SOLID------+    |
                                            +-"--+-DASH-------+--"-+
                                                 +-DOT--------+
                                                 +-DASHDOT----+
                                                 +-DASHDOTDOT-+
                                                 +-NULL-------+

   +-0-----+
>--+-------+--)-------------------------------------------------------><
   +-color-+
```

The CreatePen method creates a pen in the specified color and style. It returns the handle to a pen object. To remove the pen, use DeleteObject.

Arguments:

The arguments are:

width

The width of the lines that the pen will draw. If omitted, 1 is used as default.

style

One of the keywords listed in the syntax diagram. Values other than SOLID or NULL have no effect on pens with a width greater than 1. SOLID is the default.

color

The color number of the pen. If omitted, 0 is used as default. For a list of color numbers, refer to Definition of Terms.

Example:

The following example creates a dotted pen object with a width of 1:

```
hPen = MyDialog~CreatePen(1, "DOT", 13)
```

5.11.3. ObjectToDC

```
>>-aDialogControl~ObjectToDC(--dc--,--obj--)------------------><
```

The ObjectToDC method loads a graphic object, namely a pen or a brush, into a device context. Subsequent lines, rectangles, and arcs are drawn using the pen and brush.

Arguments:

The arguments are:

dc

The device context.

obj

The object: a pen or a brush.

Return value:

The handle of the previous active pen or brush. It can be used to restore the previous environment.

Example:

The following example activates a pen for drawing:

```
dc = MyBaseDialog~GetDC
hpen = MyDialog~CreatePen(2, "SOLID", 4)
MyDialog~ObjectToDC(dc,hpen)
... /* do lines, rectangles, ... */
MyDialog~deleteObject(hpen)
```

5.11.4. DeleteObject

```
>>-aDialogControl~DeleteObject(--obj--)------------------------><
```

The DeleteObject method deletes a graphic object, namely a pen or a brush. See CreatePen and CreateBrush for information on how to get the handle of a pen or brush.

Arguments:

The only argument is:

obj

The handle of a pen or brush.

5.12. Graphic Drawing Methods

The following methods are used to draw rectangles, lines, pixels, and arcs in a device context. See GetWindowDC, GetDC, and GetButtonDC for how to retrieve a device context. A pen and a brush can be activated using ObjectToDC before invoking the drawing methods.

Note: Because the pixel values include the title bar in a dialog it is easier to define a button filling the window, and then draw on the button.

5.12.1. Rectangle

```
>>-aDialogControl~Rectangle(--dc--,--x--,--y--,--x2--,--y2--+-----------+--)-><
                                                            +-,--"FILL"-+
```

The Rectangle method draws a rectangle to the given device context. The appearance is determined by the graphics objects currently active in the device context. The active pen draws the outline and, optionally, the active brush fills the inside area. The default pen is thin black and the default brush is white.

Arguments:

The arguments are:

dc

The device context.

x, y

The position of the upper left corner of the rectangle, in pixels.

x2, y2

The position of the lower right corner.

"FILL"

The rectangle is filled with the active brush.

Example:

The following example draws a red rectangle filled with yellow, surrounded by a black rectangle:

```
dc = self~getButtonDC(100)
brush = self~createBrush(15)        /* yellow */
pen = self~createPen(10,"solid",13) /* thick red */
oldb = self~objectToDC(dc,brush)
oldp = self~objectToDC(dc,pen)
self~Rectangle(dc, 50, 50, 200, 150, "FILL")
self~objectToDC(dc,oldp); self~deleteObject(pen)
self~objectToDC(dc,oldb); self~deleteObject(brush)
self~Rectangle(dc, 40, 40, 210, 160) /* default */
```

5.12.2. DrawLine

```
>>-aDialogControl~DrawLine(--dc--,--+-------+--,--+-------+--,--toX--,--toY--)-><
                                    +-fromX-+     +-fromY-+
```

The DrawLine method draws a line within the device context using the active pen.

Arguments:

The arguments are:

dc

The device context.

fromX, fromY

The starting position, in pixels. If omitted, the previous end point of a line or arc is used.

toX, toY

The target position.

5.12.3. DrawPixel

```
>>-aDialogControl~DrawPixel(--dc--,--x--,--y--,--color--)------><
```

The DrawPixel method draws a pixel within the device context.

Arguments:

The arguments are:

dc

The device context.

x, y

The position, in pixels.

color

The color number for the pixel. For a list of color numbers, refer to Definition of Terms.

5.12.4. GetPixel

```
>>-aDialogControl~GetPixel(--dc--,--x--,--y--)----------------><
```

The GetPixel method returns the color number of a pixel within the device context.

Arguments:

The arguments are:

dc

The device context.

x, y

The position, in pixels.

5.12.5. DrawArc

```
>>-aDialogControl~DrawArc(--dc--,--x--,--Y--,--x2--,--y2-------->

>--+---------------------------------------+--)--------------><
   +-,--startx--,--starty--,--endx--,--endy-+
```

The DrawArc method draws a circle or ellipse on the given device context using the active pen for the outline. The circle or ellipse is drawn within the boundaries of an imaginary rectangle whose coordinates are given. A partial figure can be drawn by giving the end points of two radials. By default, the figure is drawn counterclockwise, but the direction can be modified using SetArcDirection.

Arguments:

The arguments are:

dc

The device context.

x, y

The position of the upper left corner of the imaginary rectangle, in pixels.

x2, y2

The position of the lower right corner of the imaginary rectangle, in pixels.

startx, starty, endx, endy

The end points of the starting and ending radials for drawing the figure. A full circle or ellipse is drawn if no start and end are given. Omitted values default to 0. Imaginary radials are drawn from the center to the start and end points. The circle or ellipse is then drawn between the intersections of these lines with the full circle or ellipse.

Example:

This example draws a full ellipse and a quarter circle:

```
dc = self~getButtonDC(100)
pen = self~createPen(4,"solid",13)
```

```
oldp = self~objectToDC(dc,pen)
self~drawArc(dc,50,50,200,150)                /* full ellipse */
self~drawArc(dc,100,100,150,150, 200,50,75,75) /* quarter circle */
self~objectToDC(dc,oldp); self~deleteObject(pen)
```

5.12.6. GetArcDirection

```
>>-aDialogControl~GetArcDirection(--dc--)---------------------><
```

The GetArcDirection method returns the current drawing direction for the DrawArc method.

Arguments:

The only argument is:

dc

The device context.

5.12.7. SetArcDirection

```
>>-aDialogControl~SetArcDirection(--dc--,--"--+-COUNTERCLOCKWISE-+--"--)-><
                                              +-CLOCKWISE--------+
```

The SetArcDirection method changes the drawing direction for the DrawArc and DrawPie methods.

Arguments:

The arguments are:

dc

The device context.

direction

The new drawing direction.

5.12.8. DrawPie

```
>>-aDialogControl~DrawPie(--dc--,--x--,--y--,--x2--,--y2-->

>--,--startx--,--starty--,--endx--,--endy--)-------------><
```

The DrawPie method draws a pie of a circle or ellipse on the given device context using the active pen for the outline and the active brush to fill the pie. The circle or ellipse is drawn within the boundaries of an imaginary rectangle whose coordinates are given. The arc is drawn between start and end radials in the direction specified by SetArcDirection.

Arguments:

The arguments are:

dc

The device context.

x, y

The position of the upper left corner of the imaginary rectangle, in pixels.

x2, y2

The position of the lower right corner of the imaginary rectangle.

startx, starty, endx, endy

The end points of the two radials (same as for DrawArc).

5.12.9. FillDrawing

```
>>-aDialogControl~FillDrawing(--dc--,--x--,--y--,--color--)---->
```

The FillDrawing method fills an outline figure in the given device context using the active brush.

Arguments:

The arguments are:

dc

The device context.

x, y

The inside starting position for filling the outline figure with the color of the brush, in pixels.

color

The color number of the outline figure whose inside will be filled. For a list of color numbers, refer to Definition of Terms.

5.12.10. DrawAngleArc

```
>>-aDialogControl~DrawAngleArc(--dc--,--xs--,--ys--,--x--,--y-->

>--,--radius--,--startangle--,--sweepangle--)----------------><
```

The DrawAngleArc method draws a partial circle (arc) and a line connecting the start drawing point with the start of the arc on the given device context using the active pen for the outline. The circle is drawn counterclockwise with the given radius between the given angles.

Arguments:

The arguments are:

dc

The device context.

xs, ys

The start draw position, in pixels.

x, y

The center of the circle, in pixels.

radius

The radius of the circle, in pixels.

startangle, sweepangle

The starting and ending angles for the partial circle in degrees (0 is the x-axis).

Chapter 6. UserDialog Class

The UserDialog class extends the BaseDialog class. It provides methods to create a dialog with these control elements:

- Entry lines
- Push buttons
- Check boxes
- Radio buttons
- List boxes
- Combo boxes
- Frames and rectangles

Note: The class also inherits the methods of its parent class: the BaseDialog class.

Subclasses of the UserDialog can inherit the mixin AdvancedControls class. This class provides methods to dynamically create a dialog with these additional elements:

- List controls (List-view controls)
- Tree controls
- Tab controls
- Slider controls
- Progress Bar controls

There are three basic ways of creating a dialog using the UserDialog class:

- Load the dialog from a resource script using the Load method. This will create the dialog and its controls from the definitions in the resource script. An alternative to calling the `Load` method is to use the RcDialog class. This class is a subclass of the `UserDialog` class and provides a convenient way to create a dialog from a resource script. It handles the details of loading the resource script transparently and leaves the programmer free to concentrate on the details of the behavior of the dialog.
- Invoke Add... methods on an instance of this class or a subclass and create the dialog step by step, one method for one dialog item. The best place to invoke these Add... methods is to override the DefineDialog method. The DefineDialog method is called automatically when an instance of the UserDialog is created.

  In addition to creating dialog items one at a time, there are also methods that enable you to define a group of the same dialog elements together. The names of these methods begin with Add... and end with Group or Stem.
- Combine loading a dialog from a resource script and adding elements dynamically. This would define the dialog and some of the dialog controls using the resource script. The programmer could then define additional dialog controls using the Add... methods in the DefineDialog method. For instance,

this could be useful in an application that had several dialogs with similar features. The similar aspects of each dialog could be placed in a resource script and then the individual dialogs could be customized using the Add... methods.

Requires:

`UserDlg.cls` is the source file of this class.

```
::requires "oodialog.cls"
```

Subclass:

The UserDialog class is a subclass of BaseDialog.

Attributes:

Instances of the UserDialog class have the following attributes:

AktPtr

An attribute for internal use

BasePtr

An attribute for internal use

DialogItemCount

An attribute for internal use

FactorX

Horizontal size of one dialog unit (in pixels)

FactorY

Vertical size of one dialog unit (in pixels)

SizeX

The width of the dialog, in dialog units, when the underlying Windows dialog object is created.

Note: If the program creates a resizable dialog and the user changes the size of the dialog, this attribute is not updated by ooDialog. If the programmer requires that the `SizeX` attribute always reflects the actual size of the dialog, (for a resizable dialog,) then the programmer is responsible for updating its value when the size of the dialog changes. The programmer would use the ConnectResize method to receive notifications when the dialog changed size and update the attribute accordingly.

SizeY

The height of the dialog, in dialog units, when the Windows dialog object is created.

Note: In a similar fashion to the `SizeX` attribute, the programmer would be responsible for updating this attribute for resizable dialogs.

Methods:

Instances of the UserDialog class implement the methods listed in the following table.

Table 6-1. UserDialog Instance Methods

Method...	...on page
AddBitmapButton	AddBitmapButton
AddBlackFrame	AddBlackFrame
AddBlackRect	AddBlackRect
AddButton	AddButton
AddButtonGroup	AddButtonGroup
AddCheckBox	AddCheckBox
AddCheckBoxStem	AddCheckBoxStem
AddCheckGroup	AddCheckGroup
AddComboBox	AddComboBox
AddComboInput	AddComboInput
AddEntryLine	AddEntryLine
AddGrayFrame	AddGreyFrame
AddGrayRect	AddGrayRect
AddGroupBox	AddGroupBox
AddIcon	AddIcon
AddInput	AddInput
AddInputGroup	AddInputGroup
AddInputStem	AddInputStem
AddListBox	AddListBox
AddMenuItem	AddMenuItem
AddMenuSeparator	AddMenuSeparator
AddOkCancelLeftBottom	AddOkCancelLeftBottom
AddOkCancelLeftTop	AddOkCancelLeftTop
AddOkCancelRightBottom	AddOkCancelRightBottom
AddOkCancelRightTop	AddOkCancelRightTop
AddPasswordLine	AddPasswordLine
AddPopupMenu	AddPopupMenu
AddRadioButton	AddRadioButton
AddRadioGroup	AddRadioGroup
AddRadioStem	AddRadioStem
AddScrollBar	AddScrollBar
AddText	AddText
AddWhiteFrame	AddWhiteFrame
AddWhiteRect	AddWhiteRect

Method...	...on page
Create	Create
CreateCenter	CreateCenter
CreateMenu	CreateMenu
DefineDialog	DefineDialog
Init	Init
InitAutoDetection	InitAutoDetection
Load	Load
LoadFrame	LoadFrame
LoadItems	LoadItems
LoadMenu	LoadMenu
SetMenu	SetMenu
StartIt	StartIt
StopIt	StopIt

6.1. Init

```
>>-aUserDialog~Init(--+----------+--+---------------+--)------><
                      +-DlgData.-+  +-,--headerFile-+
```

The Init method initializes a new dialog object.

Arguments:

The arguments are:

DlgData.

A stem variable that is used to initialize the data fields of the dialog. If the dialog is terminated by means of the OK button, the values of the dialog's data fields are copied to this variable. The ID of the dialog items is used to name each entry within the stem.

headerFile

A file defining symbolic IDs for dialog resources. Symbolic IDs defined in the file can be used in method calls where resource IDs are required.

Example:

The following example creates a new dialog object, adding any symbolic IDs defined in `resources.h` to the object's ConstDir directory:

```
MyDialog=.UserDialog~new(aStem., "resources.h")
```

The example for the `init` method in the BaseDialog section shows the use of the dialog data stem and header file arguments in more detail.

6.2. InitAutoDetection

```
>>-aUserDialog~InitAutoDetection------------------------------><
```

The InitAutoDetection method is called by the Init to determine whether or not automatic data field detection should be used. For a UserDialog, autodetection is disabled.

Protected:

This method is protected. It is called by the class itself and can be overwritten.

Example:

The following example overrides the method to switch off auto detection:

```
::class MyClass subclass UserDialog

::method InitAutoDetection
self~NoAutoDetection
```

6.3. Create

```
>>-aUserDialog~Create(--x--,--y--,--cx--,--cy--,--title--,------>

>--+---------------------------+--,--+----------+--------------->
   |    .-----------------.    |     +-dlgClass-+
   |    V                 |    |
   +-"----+-NOMENU------+-+--"-+
          +-VISIBLE-----+
          +-NOTMODAL----+
          +-SYSTEMMODAL-+
          +-THICKFRAME--+
          +-MINIMIZEBOX-+
          +-MAXIMIZEBOX-+

>--,--+----------+--,--+----------+--,--+----------+--)-------><
      +-fontName-+     +-fontSize-+     +-expected-+
```

The Create method creates the Windows dialog you previously defined with Add... methods. You can set the size, title, and style of the dialog. If the return code of Create is 0 the dialog creation failed and you should not try to execute the dialog object.

Arguments:

The arguments are:

x, y

The position of the upper-left edge of the dialog in dialog units

cx, cy

The extent (width and height) of the dialog in dialog units

title

The dialog's title that is displayed in the title bar

options

One or more of the keywords listed in the syntax diagram, separated by blanks:

NOMENU

Creates a dialog without a system menu

VISIBLE

Creates a visible dialog

NOTMODAL

Creates a dialog with a normal window frame

SYSTEMMODAL

Creates a dialog that blocks all other windows

THICKFRAME

Creates a dialog with a thick frame

MINIMIZEBOX

Creates a dialog with a minimize button.

MAXIMIZEBOX

Creates a dialog with a maximize button.

dlgClass

Name of the window class used for the dialog. This argument must be omitted or an empty string.

fontName

The name of a font used by the dialog for all text. The default font is System.

fontSize

The size of the font used by the dialog. The default value is 8.

expected

This argument determines the maximum number of dialog elements (static text, entry lines, list boxes, and the like) the dialog can handle. The default value is 200. If your dialog has more than 200 elements, you must set this value; otherwise, the dialog fails.

Example:

The following example creates a dialog with a size of 300 by 200 dialog units. The dialog has no system menu in its upper-left corner. It has a thick frame (therefore the dialog will get resizable) and a 12-point font. The dialog has capabilities for up to 100 elements.

```
rc = MyDialog~Create(20, 20, 300, 200, "My first Dialog", ,
  "THICKFRAME NOMENU", , "Courier", 12, 100)
if rc <> 0 then MyDialog~Execute
```

6.4. CreateCenter

```
>>-aUserDialog~CreateCenter(--cx--,--cy--,--title--,------------>

>--+---------------------------+--,--+----------+--------------->
   |    .-----------------.    |     +-dlgClass-+
   |    V                 |    |
   +-"----+-NOMENU------+-+--"-+
          +-VISIBLE-----+
          +-NOTMODAL----+
          +-SYSTEMMODAL-+
          +-THICKFRAME--+
          +-MINIMIZEBOX-+
          +-MAXIMIZEBOX-+

>--+-----------------------------------------------+--)-------><
   +-,--fontName--+-----------------------------+-+
                  +-,--fontSize--+-------------+-+
                                 +-,--expected-+
```

The CreateCenter method creates a dialog and centers its position. See Create for a description of all arguments and an example.

6.5. DefineDialog

```
>>-aUserDialog~DefineDialog----------------------------------><
```

The DefineDialog method is called by Create. It is designed to be overwritten in a subclass of UserDialog. You should do all the definition of dynamically added dialog controls within this method, using the Add... Methods.

Protected:

This method is protected. There is no need to call this method from anywhere else than Create.

Example:

When the dialog is created, a push button and an entry line are added to its client area:

```
::class MyDialog subclass UserDialog
     .
     .
     .
::method DefineDialog
self~AddButton(401, 20, 235, 40, 15, "&More...")
self~AddEntryLine(402, INPUT_1, 20, 170, 150, 10)
     .
     .
     .
```

6.6. Load

```
>>-aUserDialog~Load(--resourceFileName--,--dialogId------------->

>--+-------------------------------------------------------+----->
   +-,--+-----------------------------+--+-------------+-+
        |    .--------------------.    |  +-,--expected-+
        |    V                    |    |
        +-"----+-CENTER---------+-+--"-+
               +-CONNECTBUTTONS-+
               +-CONNECTRADIOS--+
               +-CONNECTCHECKS--+

>--)----------------------------------------------------------><
```

The `Load` method creates the dialog based on the data of a given resource script (a file with the extension .RC). It calls the LoadFrame and LoadItems methods to retrieve the dialog data from the file.

Return code:

The return code is 0 for a successful load and 1 otherwise

Arguments:

The arguments are:

resourceFileName

The name of the resource script of the dialog

dialogId

The ID (number) of the dialog. Note that each dialog has a unique ID assigned to it. There can be more than one dialog definition in one resource file. If there is only one dialog resource in the resource file, you do not have to indicate the ID.

options

One or more of the keywords listed in the syntax diagram, separated by blanks:

CENTER

The dialog is positioned in the center.

CONNECTBUTTONS

For each button a connection to an object method is established automatically. See ConnectControl for a description of connecting buttons to a method.

CONNECTRADIOS

Similar to CONNECTBUTTONS, this option enforces the method to connect the radio buttons.

CONNECTCHECKS

This option connects the check box controls.

expected

This is the maximum number of dialog elements the dialog object can handle. See Create.

Example:

The following example creates a dialog based on the values for dialog `100` in `Dialog1.rc`. It also connects the push and radio buttons to a message named after the buttons' title.

```
MyDlg = .UserDialog~new()
MyDlg~Load("Dialog1.rc", 100, "CONNECTBUTTONS CONNECTRADIOS")
```

6.7. LoadFrame

```
>>-aUserDialog~LoadFrame(--resfile----------------------------->

>--+-------------------------------------------------------+---->
   +-,--+----------+--+-----------------------------------+-+
        +-dialogid-+  +-,--+----------+--+-------------+-+
                           +-"CENTER"-+  +-,--expected-+

>--)---------------------------------------------------------><
```

The LoadFrame method creates the window frame using the data of the given dialog resource with dialogid in file resfile. It is usually called by the Load method.

Protected:

This method is protected. It can only be used internally within a class method.

Arguments:

The arguments are:

resfile

The name of the resource file

dialogid

The ID of the dialog. It can be omitted if there is just one dialog; otherwise it has to be specified.

expected

The number of expected dialog items

Example:

The following example overrides the Load method, so it loads the dialog window (just the frame) but not its contents:

```
::class WindowOnlyDialog subclass UserDialog
     .
     .
     .
::method Load
self~LoadFrame("Dialog2.rc", 100, "CENTER", 20)
```

6.8. LoadItems

```
>>-aUserDialog~LoadItems(--resFile----------------------------->

>--+-------------------------------------------------------+----->
   +-,--+----------+--+--------------------------------+-+
        +-dialogId-+  |       .--------------------.    |
                      |       V                    |    |
                      +-,--"----+-CONNECTBUTTONS-+-+--"-+
                                +-CONNECTRADIOS--+
                                +-CONNECTCHECKS--+

>--)---------------------------------------------------------><
```

The LoadItems method creates the dialog items, using the data of the given resource script. It is either called by the Load method, or it can be used in the context of a category dialog.

Protected:

This method cannot be called from outside the class.

Arguments:

See Load for a description.

Example:

In the following example the dialog is created either with the items of dialog 200 or dialog 300, depending on the argument:

```
::class MyDialog subclass UserDialog
      .
      .
      .
::method Load
   use arg view
   self~LoadFrame("Dialog2.rc", 200, "CENTER", 200)
   if view="special" then
      self~LoadItems("Dialog2.rc", 200, "CONNECTBUTTONS")
   else
      self~LoadItems("Dialog2.rc", 300, "CONNECTBUTTONS")
```

6.9. Add... Methods

The methods listed below (all starting with Add) can be used to create a dialog dynamically without any resource script (.RC file). The methods are used to define the individual dialog controls. The methods can also be used in a dialog that is created by loading a resource script. The methods will then be used to define controls in addition to the ones defined in the resource script.

The recommended way to create a dialog dynamically is to subclass from UserDialog and put all Add... statements into the DefineDialog method, which is executed when the dialog is about to be created.

The Add... methods call the matching Connect... methods to create the associated Object Rexx attribute. Add... methods cannot be used after Execute has started.

> **Note:** Unless otherwise stated, the coordinates (the x, y, cx, and cy arguments) of the dialog control definitions are specified in dialog units.

The behavior and appearance of all dialog controls created using one of the Add... methods are controlled by a style option. For each of the Add... methods, one of the arguments consists of a set of keywords that define the style of the control being defined. These keywords allow the programmer to change the default behavior and / or appearance of the control.

There are five styles that all dialog controls share. The following list explains these styles and the keywords that are used for the styles. In many cases, there is a common default value for the style and the programmer only needs to specify a keyword when she wants a non-default value for the style. For instance, normally all dialog controls are created enabled. The programmer would only need to specify the **DISABLED** keyword when the control should be disabled as the dialog is first displayed.

Visible

All windows (and every dialog control is a window) can be visible or invisible. When a control is invisible it is not displayed on the screen and the user will not see it. The control is also not active and will not accept any keyboard or mouse input. But, the control still exists and does not lose any of its internal state. Since the control is invisible, whatever is underneath it will be displayed on the screen. Normally this would be the owner dialog. However, it could also be another dialog control. If this other dialog control is visible, the user will see and can interact with it rather than the invisible control.

In ooDialog by default all controls created using the Add... methods will be created visible. To change the default for any control when using the Add... methods add the **HIDDEN** keyword to the style argument. This will make the control invisible when the dialog is first shown on the screen.

For any control, the visible style can be changed during program execution by using the Hide or Show methods. (Or any of the related methods like HideFast, Display, etc..)

Enabled

All windows can be disabled or enabled. When a dialog control is disabled it will not accept keyboard or mouse input and therefore the user can not interact with it. The operating system will draw disabled controls in a different manner than enabled controls to give the user a visual clue that the control is disabled.

When using the Add... methods controls will be created enabled. To change this for any control add the **DISABLED** keyword to the style argument. The control will then be disabled when the dialog is first shown on the screen.

All controls can have their enabled style changed during program execution by using the Disable or Enable methods. (Or any of the related methods like DisableItem, EnableItem, etc..)

Border

Most, but not all, windows are drawn with a border. When using the Add... methods individual controls are drawn by default with, or without, a border. The default is changed by using either the **BORDER** or **NOBORDER** keywords. The reference for each individual Add... method notes the default for the control for this style and which keyword is used to change the default. In general, this style can not be changed after a control is created.

Group

This style is used to place a series of consecutive dialog controls in a group. The first control with the group style starts a new group of controls. This group continues for each control that does not have the group style. As soon as a successive control is encountered with the group style, a new group is started. This style is only meaningful for dialog control windows.

The Windows dialog manager uses this style to determine the behavior of a dialog, mostly the navigation behavior. Within a group of controls the user navigates from one control to the other using the arrow keys rather than the tab key.

The **GROUP** keyword is used to give a control the group style when it is created using one of the Add... methods. After the control has been created its group style can be changed during the execution of a dialog by using the SetGroup method of the PlainBaseDialog class or the Group method of the DialogControl class.

Tabstop

The tabstop style is another style that is only meaningful with dialog controls. When a control has the tabstop style the user can navigate to it using the tab key. If a control does not have this style, the tab key will skip over the control. Dialog controls in the same class usually all have the same style for tabstop. As an example, all button controls usually have the tabstop style, whereas all static text controls usually do not have the tabstop style.

When a control as added to an UserDialog instance it is given the tabstop style that is normal for the control. The individual Add... methods document the default for the control. The programmer uses either the **TAB** or **NOTAB** keywords to change the default. After controls have been created they can have their tabstop style changed using the SetTabStop method of the PlainBaseDialog class or the TabStop method of the DialogControl class.

6.9.1. AddButton

```
>>-aUserDialog~AddButton(--id--,--x--,--y--,--cx--,--cy--,--text-->

>--+-----------------------------------------------+--)--------><
   +-,--msgToRaise--+---------------------------+-+
                    |       .---------------+    |
                    |       V               |    |
                    +-,--"----+-DEFAULT---+-+--"-+
                              +-OWNER-----+
                              +-LEFT------+
                              +-RIGHT-----+
                              +-TOP-------+
                              +-BOTTOM----+
                              +-MULTILINE-+
                              +-NOTIFY----+
                              +-FLAT------+
                              +-HIDDEN----+
                              +-DISABLED--+
                              +-GROUP-----+
                              +-NOTAB-----+
                              +-BORDER----+
```

The AddButton method adds a push button to the dialog and connects it with a method that is invoked whenever the button is clicked.

Arguments:

The arguments are:

id

The resource ID (numeric or symbolic) for the button. Each control in a dialog (other than static controls) must be assigned a unique resource ID.

x, y

The position of the button's upper-left corner relative to the dialog measured in dialog units

cx

The width of the button in dialog units

cy

The height of the button in dialog units

text

The label that is displayed for the button

msgToRaise

The name of a method of the dialog class that is to be invoked each time the button is clicked

style

The last argument can be one or more of the following keywords:

DEFAULT

The button becomes the default button in the dialog.

OWNER

The programmer is completely responsible for drawing the button when the dialog receives the WM_DRAWITEM message. Currently this would be difficult (but not impossible) to implement in ooDialog. A simpler approach for an owner drawn button is to use a bitmap button. See the AddBitmapButton method.

LEFT

Left justifies the text in the button rectangle. If neither LEFT nor RIGHT are specified the text is centered in the button rectangle.

RIGHT

Right justifies the text in the button rectangle. If neither LEFT nor RIGHT are specified the text is centered in the button rectangle.

TOP

The text is aligned at the top of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

BOTTOM

The text is aligned at the bottom of the button rectangle. If neither TOP nor BOTTOM are specified the text is vertically centered in the button rectangle.

MULTILINE

If the text for the label of the button is longer than the width of the button rectangle, the operating system will word wrap the text producing more than one line. The height of the button rectangle has to be sufficient to display the extra line(s) or the text is clipped.

NOTIFY

Enables the button to send notifications for the gained focus, lost focus, and double click events. This is only necessary when the ConnectButtonNotify method is used, and only for the GOTFOCUS, LOSTFOCUS, or DBLCLK keywords of that method.

FLAT

The button is drawn with a flat appearance, i.e., it is drawn as a flat rectangle the label text inside the rectangle.

HIDDEN

Create the button without the visible style.

DISABLED

Create the button without the enabled style.

GROUP

Create the button with the group style.

NOTAB

Create the button without the tabstop style. By default the button is created with the tabstop style.

BORDER

Create the button with the border style. By default buttons are created without a border.

Example:

The following example creates a push button entitled "Get new Info" at position x=100, y=80 and size width=40, height=15. The button's ID is 555, and if the button is clicked, the getInfo message is sent to the dialog object. The button is created without the tabstop style. If the user navigates the dialog using the keyboard, the tab key will skip over the button.

```
MyDialog~AddButton(555, 100, 80, 40, 15, "&Get new Info", ,
"getInfo", "NOTAB")
```

6.9.2. AddBitmapButton

```
>>-aUserDialog~AddBitmapButton(--id--,--x--,--y--,--+----+--,--->
                                                    +-cx-+

>--+----+--,--+------+--,--+------------+--,--bmpNormal--,------>
```

```
   +-cy-+     +-text-+     +-msgToRaise-+

>--+------------+--,--+-------------+--,--+-------------+------->
   +-bmpFocused-+     +-bmpSelected-+     +-bmpDisabled-+

>--+---------------------------+--)---------------------------><
   |       .--------------.    |
   |       V              |    |
   +-,--"----+-DEFAULT--+-+--"-+
             +-FRAME----+
             +-USEPAL---+
             +-INMEMORY-+
             +-STRETCH--+
             +-NOTIFY---+
             +-HIDDEN---+
             +-DISABLED-+
             +-GROUP----+
             +-NOTAB----+
             +-BORDER---+
```

The AddBitmapButton method adds a push button with a bitmap (instead of plain text) to the dialog. You can provide four different bitmaps representing the four states of a button.

The bitmaps can be specified by either a file name or a bitmap handle. You can retrieve a bitmap handle by loading a bitmap stored in a file into memory, using the method LoadBitmap). If you pass a bitmap handle to the method, you must use the INMEMORY option.

Arguments:

The arguments are:

id

The resource ID (numeric or symbolic) for the button. Each control in a dialog (other than static controls) must be assigned a unique resource ID.

x, y

The position of the button's upper-left corner relative to the dialog measured in dialog units

cx

The width of the button in dialog units. If you omit this argument the width of the button will be set to 0 and you will not be able to see the button.

cy

The height of the button in dialog units. If you omit this argument the height of the button will be set to 0 and you will not be able to see the button.

text

The text label for the button. Although the user will not see this text, (they see the bitmap instead,) the text is still associated with the button and can be retrieved by the programmer using the Title method of the `DialogControl` class.

msgToRaise

The name of a method of the dialog class that is to be invoked each time the button is clicked.

bmpNormal

A bitmap that is displayed when the button is in the normal state. I.e., it is not focused, not selected, and not disabled. This argument must be specified. If any of the other three bitmap arguments are omitted, this bitmap will be used in its place.

bmpFocused

A bitmap that is displayed if the button is focused. When a button is focused it receives the keyboard input and can be clicked by using the Enter or Space keys. Normally the focused button is surrounded by a dashed frame. If this argument is ommitted, the `bmpNormal` bitmap is displayed for the focused state.

bmpSelected

A bitmap that is displayed while the button is clicked and held down. If this argument is ommitted, the `bmpNormal` bitmap is displayed for the selected state.

bmpDisabled

A bitmap that is displayed if the button is disabled. If this argument is ommitted, the `bmpNormal` bitmap is displayed for the disabled state.

style

The last argument can be one or more of the following keywords:

DEFAULT

The button becomes the default button in the dialog.

FRAME

The button has a 3D frame. This gives your bitmap the same behavior as a standard Windows button.

USEPAL

The color palette of the bitmap is loaded and used. This argument should be specified for just one of the dialog buttons, because only one color palette can be active at any time.

INMEMORY

Specifies that the bitmap was loaded into memory before. If you switch often between different bitmaps within one button, the loading of all bitmaps into memory increases performance.

STRETCH

If this option is specified and the extent of the bitmap is smaller than the extent of the button rectangle, the bitmap is adapted to match the extent of the button.

NOTIFY

Enables the button to send notifications for the gained focus and lost focus events. This is only necessary when the ConnectButtonNotify method is used, and only for the GOTFOCUS and LOSTFOCUS keywords of that method. Note that this style is not needed for bitmap buttons to receive the double click (DBLCLK) event as it is for other buttons.

HIDDEN

Create the button without the visible style.

DISABLED

Create the button without the enabled style.

GROUP

Create the button with the group style.

NOTAB

Create the button without the tabstop style. By default the button is created with the tabstop style except in these conditions: the bmpFocused is ommitted, and the bmpSelected is ommitted, and the FRAME style keyword is not used.

TAB

Create the button with the tabstop style. The button is normally created with the tabstop style and this keyword is not necessary. However, if the bmpFocused argument is ommitted, and the bmpSelected argument is ommitted, and the FRAME style keyword is not used, the button is created without the tabstop style. In this case only, it is necessary to use the TAB keyword to change the default behavior.

BORDER

Create the button with the border style. By default the button is created without a border.

Example:

The following example defines a button with ID 601. The bitmap in the `Button1.bmp` file is displayed for the push button instead of a black text on a grey background. If the button is disabled (by using the DisableItem method, see page DisableItem), the bitmap is exchanged and Button1D.bmp is shown instead. If the button is clicked, the BmpPushed message is sent.

```
MyDialog~AddBitmapButton(601, 20, 317, 80, 30, , "BmpPushed", ,
                         "Button1.bmp", , ,"Button1D.bmp","FRAME USEPAL")
```

6.9.3. AddIcon

```
>>-aUserDialog~AddIcon(--id--,--fileName--)-------------------><
```

The AddIcon method adds an icon resource to the dialog. This is equivalent to using the `ICON` statement in a resource script.

Arguments:

The arguments are:

id

The resource ID for the icon.

fileName

The file name of the icon. If the icon file is not in the current directory, then a relative or fully qualified file name should be used.

Example:

The following example creates a very simple dialog that has an OK button and a static text box. The `addIcon` method is used to load an icon resource using the ID of 15. That icon is then used in the `execute` method for the dialog icon. (The dialog icon is the icon shown in the title bar of the dialog.)

```
  dlg = .SimpleDialog~new
  if dlg~initCode = 0 then do
    dlg~create(30, 30, 150, 100, "The Simple Dialog", "VISIBLE")
    dlg~execute("SHOWTOP", 15)
    dlg~deinstall
  end

::requires "OODWin32.cls"

::class SimpleDialog subclass UserDialog

::method defineDialog

  self~addText(10, 20, 130, 30, "Simple message", "CENTER BORDER", 100)
  self~addButton(IDOK, 105, 70, 35, 15, "OK")
  self~addIcon(15, "MyAppIcon.ico")
```

6.9.4. AddGroupBox

```
>>-aUserDialog~AddGroupBox(--x--,--y--,--cx--,--cy--,--text----->

>--+------------------------+--)------------------------------><
   +-,--"BORDER"--+-------+-+
                  +-,--id-+
```

The AddGroupBox method adds a group box to the dialog. A group box has a frame and a title.

Arguments:

The arguments are the same as for AddButton, with the changes listed below:

text

The title of the group box

options

BORDER

A rectangle is drawn around the group box

6.9.5. AddText

```
>>-aUserDialog~AddText(--x--,--y--,--+----+--,--+----+--,--text-->
                                     +-cx-+     +-cy-+

>--+----------------------------------------+--)---------------><
   +-,--+----------------------+--+-------+-+
        |    .------------.    |  +-,--id-+
        |    V            |    |
        +-"----+-HIDDEN-+-+--"-+
               +-SIMPLE-+
               +-RIGHT--+
               +-CENTER-+
               +-BORDER-+
```

The AddText method adds a static text element to the dialog.

Arguments:

The arguments are the same as for AddButton, with the changes listed below:

text

The text string to be displayed

options

This argument can be one or more of:

HIDDEN

The text is not visible at startup time

SIMPLE

Simple text field

RIGHT

The text is aligned to the right

CENTER

The text is centered. If neither RIGHT or CENTER is specified, the text is aligned to the left.

BORDER

A rectangle is drawn around the text

If not specified RIGHT or CENTER, the text is aligned to the left.

id

If omitted, ID -1 is used.

6.9.6. AddEntryLine

```
>>-aUserDialog~AddEntryLine(--id--,--+------+--,--x--,--y--,--cx-->
                                     +-name-+

>--+-----------------------------------------------+---)---------><
   +-,--+----+--+-------------------------------+-+
        +-cy-+  |       +-------------------+    |
                |       V                   |    |
                +-,--"----+-AUTOSCROLLH---+-+--"-+
                          +-HSCROLL-------+
                          +-AUTOSCROLLV---+
                          +-VSCROLL-------+
                          +-MULTILINE-----+
                          +-HIDESELECTION-+
                          +-NOWANTRETURN--+
                          +-READONLY------+
                          +-KEEPSELECTION-+
                          +-PASSWORD------+
                          +-CENTER--------+
                          +-RIGHT---------+
                          +-UPPER---------+
                          +-LOWER---------+
                          +-NUMBER--------+
                          +-OEM-----------+
                          +-HIDDEN--------+
                          +-DISABLED------+
                          +-GROUP---------+
                          +-NOTAB---------+
                          +-NOBORDER------+
```

The AddEntryLine method adds an entry line (an Edit Control in Windows terms) to the dialog.

Arguments:

The arguments are:

id

The resource ID (numeric or symbolic) for the edit control. Each control in a dialog (other than static controls) must be assigned a unique resource ID.

name

This is the name of the edit control. An attribute with exactly this name is added to the object and provides data for the dialog item automatically. See ConnectEntryLine.

x, y

The position of the upper-left corner relative to the dialog's client area measured in dialog units

cx

The length of the edit control in dialog units

cy

The height of the edit control. If this argument is omitted or equal to 0, the height is calculated to fit the font's height.

style

The last argument can be one or more of the following style keywords. (Each keyword is separated by a blank.):

Note

The behavior described here for the style keywords may vary slightly depending on the version of Windows.

AUTOSCROLLH

Automatically scrolls text to the right when the user types a character at the end of the line. In a single-line edit control text can not be added past the width of the control without this style. Multi-line edit controls do not need this style if a horizontal scroll bar is added.

HSCROLL

Adds a horizontal scroll bar to the bottom of the edit control. A horizontal scroll bar is only functional in multi-line edit controls.

AUTOSCROLLV

Automatically scrolls text up when the user presses ENTER on the last line. This style only effects multi-line edit controls and is not needed if a vertical scroll bar is added.

VSCROLL

Adds a vertical scroll bar to the right of the edit control.

MULTILINE

Designates a multiple-line edit control. (The default is single line.) Multi-line edit controls have the WANTRETURN and KEEPSELECTION styles unless the NOWANTRETURN and / or HIDESELECTION keywords are specified.

When a multi-line edit control has the WANTRETURN style, pressing the ENTER key inserts a line break in the text of the control. Without the WANTRETURN style, pressing the ENTER key clicks the default button. With the KEEPSELECTION style, any selected text in the edit control keeps its highlight when the control loses focus.

HIDESELECTION

Removes the KEEPSELECTION style from multi-line edit controls. This keyword only effects multi-line edit controls.

NOWANTRETURN

Removes the WANTRETURN style from multi-line edit controls. This keyword only effects multi-line edit controls.

READONLY

Prevents the user from entering or editing text in the edit control.

KEEPSELECTION

Selected text in an edit control keeps its highlight when the focus is shifted away from the control.

PASSWORD

Characters typed into the edit control are masked. Rather than displaying the character typed, the edit control displays the mask character. Prior to Windows XP the default mask character was the asterisk. In Windows XP the default mask character is a black circle. The default password character can be changed by using the PasswordChar= method.

CENTER

Centers text in an edit control. (In Windows 95 and NT this style only effects multi-line edit controls.)

RIGHT

Aligns text flush right in an edit control. (In Windows 95 and NT this style only effects multi-line edit controls.)

UPPER

Converts all characters to uppercase as they are typed into the edit control.

LOWER

Converts all characters to lowercase as they are typed into the edit control.

NUMBER

Only allows digits to be typed into the edit control. If the user types an non-digit character an information balloon is displayed informing the user that only digits are allowed. This style does not prevent the user from pasting non-digits into the control.

OEM

Converts text entered in the edit control from the Windows character set to the OEM character set and then back again to the Windows character set. This ensures proper character conversion in certain circumstances. Microsoft states that this style is most useful when edit controls contain file names that will be used with file systems that do not support Unicode. For further details see the Windows documentation provided by Microsoft.

HIDDEN

Create the edit control without the visible style.

DISABLED

Create the edit control without the enabled style.

GROUP

Create the edit control with the group style.

NOTAB

Create the edit control without the tabstop style. By default the edit control is created with the tabstop style.

NOBORDER

Create the edit control without the border style. By default edit controls are created with a border.

Example:

The following example puts the edit control with ID 201 and length of 150 dialog units close to the upper-left corner of the dialog's client area. The FIRSTNAME attribute is created and connected to the dialog item. If the entered data is longer than 150 dialog units, the edit control scrolls horizontally.

```
MyDialog~AddEntryLine(201, "FIRSTNAME", 12, 14, 150, ,"AUTOSCROLLH")
```

6.9.7. AddPasswordLine

```
>>-aUserDialog~AddPasswordLine(--id--,--+------+--,--x--,--y--,--cx-->
                                        +-name-+

>--+---------------------------+-- )-------------------------------><
   +-,--+----+--+------------+-+
        +-cy-+  +-,--style---+
```

The AddPasswordLine method adds a password edit control that does not echo the characters entered but displays the password character instead. This method is a convenience method and is used exactly like

Arguments:

See AddEntryLine for a description of the arguments.

6.9.8. AddListBox

```
>>-aUserDialog~AddListBox(--id--,--+------+--,--x--,--y--,--cx--,-->
                                   +-name-+

>--cy--+---------------------------+-- )----------------------><
       |       +--------------+    |
       |       V              |    |
       +-,--"----+-MULTI----+-+--"-+
                 +-NOTIFY---+
                 +-SORT-----+
                 +-COLUMNS--+
                 +-VSCROLL--+
                 +-HSCROLL--+
                 +-NOBORDER-+
                 +-MCOLUMN--+
                 +-PARTIAL--+
                 +-SBALWAYS-+
```

Adds a list box to the dialog.

Arguments:

The arguments are the same as for AddEntryLine, with the changes listed below:

Options

The last argument can be one or more of:

MULTI

Makes the list box a multiple choice list box, that is, you can select more than one line.

NOTIFY

A message is posted whenever the user selects an item of the list box. To use this feature you have to connect the list to a method (see ConnectList).

SORT

The items in the dialog are listed in the noted order.

COLUMNS

The list box can handle tab characters ("09"x). Use this option together with the SetListTabulators method (see page SetListTabulators) to have more than one column in a list.

VSCROLL

Adds a vertical scroll bar to the list box. Scroll bars appear only if the list contains more lines than can fit in the available space.

HSCROLL

Adds a horizontal scroll bar to the list box. See also SetListWidth.

NOBORDER

Draw list without drawing a rectangle around it.

MCOLUMN

Makes the list box a multicolumn list box that can be scrolled horizontally. SetListColumnWidth sets the width of the columns.

PARTIAL

The size of the list box equals the size specified by the application when it created the list box. Windows usually sizes a list box such that the list box does not display partial items.

SBALWAYS

The list box shows a disabled scroll bar if there is no need to scroll. If you do not specify this option, the scroll bar is hidden when the list box does not contain enough items.

Note: A list box does not support a horizontal scroll bar.

6.9.9. AddComboBox

```
>>-aUserDialog~AddComboBox(--id--,--+------+--,--x--,--y--,--cx--,--cy-->
                                    +-name-+

>--+--------------------------+--)-----------------------------------><
   |       +-------------+    |
```

```
|       V             |    |
+-,--"----+-SIMPLE--+-+--"-+
          +-LIST----+
          +-SORT----+
          +-VSCROLL-+
```

The AddComboBox method adds a combo box to the dialog. A combo box is a combination of an entry line and a list box.

Arguments:

The arguments are the same as for AddEntryLine, with the changes listed below:

options

The last argument can be one or more of:

SIMPLE

Displays the list box all the time.

LIST

No free text can be entered in the entry line; the list contains selectable items only.

SORT

The items in the list are sorted by the combo box itself.

VSCROLL

Adds a vertical scroll bar to the combo box.

6.9.10. AddCheckBox

```
>>-aUserDialog~AddCheckBox(--id--,--+------+--,--x--,--y--,--+----+--,-->
                                    +-name-+                +-cx-+

>--+----+--,--text--+-----------------------+--)---------------------><
   +-cy-+           |       +-----------+    |
                    |       V           |    |
                    +-,--"------GROUP---+--"-+
```

The AddCheckBox method adds a check box to the dialog.

Arguments:

The arguments are the same as for AddEntryLine, with the changes listed below:

name

The name of the check box. If omitted, text is used.

text

The text that is displayed next to the check box.

6.9.11. AddRadioButton

```
>>-aUserDialog~AddRadioButton(--id--,--+------+--,--x--,--y--,--cx-->
                                       +-name-+

>--,--cy--,--text--+----------------+--)--------------------------><
                   +-,--"--GROUP--"-+
```

The AddRadioButton method adds a radio button to the dialog.

Arguments:

The arguments are the same as for AddEntryLine, with the changes listed below:

name

The name of the radio button

text

The text that is displayed next to the radio button

options

Valid values for the last argument are:

GROUP

Makes the radio button the beginning of a new group. Use this option just for the first radio button if you want to make all radio buttons dependent. In each group if you select a radio button, the previously selected button is automatically deselected.

Example:

The following example defines seven radio buttons with IDs 501 through 507:

```
RText.1="Monday"
RText.2="Tuesday"
RText.3="Wednesday"
RText.4="Thursday"
```

```
RText.5="Friday"
RText.6="Saturday"
RText.7="Sunday"

do i=1 to 7
 MyDialog~AddRadioButton(500+i, , 20, i*15+13, 40, 14, RText.i)
end
```

Note: There are also methods that create a whole group automatically (see the AddRadioGroup method below and AddRadioStem).

6.9.12. AddRadioGroup

```
>>-aUserDialog~AddRadioGroup(--startId--,--x--,--y--,--+----+--->
                                                       +-cx-+

         +------+
         V      |
>--,--"----text-+--"--+-----------------------------------+--)-->
                      +-,--+------------+--+-----------+-+
                           +-"NOBORDER"-+  +-,--idstat-+
```

The AddRadioGroup method creates a group of radio buttons.

Arguments:

The arguments are:

startId

The ID of the first radio button. The startId is increased by 1 for each additional radio button and then assigned to the dialog item.

x, y

The position of the first radio button control. The other radio buttons are positioned automatically.

cx

The length of the radio button plus text. If omitted, the space needed is calculated.

text

The text string for each radio button. Single strings have to be separated by blank spaces. This argument determines the number of radio buttons in total.

options

The only option is NOBORDER, which prevents the method from placing a group box around the group.

idstat

This argument is used to set the static frame ID.

Example:

The following example adds a group of three radio buttons with IDs 301, 302, and 303 to the dialog (see Sample Radio Button Group):

```
MyDialog = .UserDialog~new
MyDialog~Create(100,100,80,60,"Radio Button Group")
MyDialog~AddRadioGroup(301, 23, 18, ,"Fast Medium Slow")
MyDialog~fast = 1
MyDialog~Execute
```

Figure 6-1. Sample Radio Button Group

6.9.13. AddCheckGroup

```
>>-aUserDialog~AddCheckGroup(--startId--,--x--,--y--,--+----+--,-->
                                                       +-cx-+

      +------+
      V      |
>--"----text-+--"--+-----------------------------------+-- )---->&lt;
                   +-,--+------------+--+-----------+-+
                        +-"NOBORDER"-+  +-,--idstat-+
```

The AddCheckGroup method creates a group of check boxes. See AddRadioGroup for a full description.

Example:

The following example adds a group with four check boxes to the dialog. Two check boxes are preselected (see Sample Check Box Group):

```
MyDialog~AddCheckGroup(401, 23, 18, ,"Smalltalk C++ ObjectRexx OO-COBOL")
```

```
MyDialog~smalltalk = 1
MyDialog~objectrexx = 1
```

Figure 6-2. Sample Check Box Group

6.9.14. AddInput

```
>>-aUserDialog~AddInput(--id--,--+----------+--,--x--,--y--,---->
                                 +-attrName-+

>--+-----+--,--cx2--,--+----+--,--text------------------------->
   +-cx1-+             +-cy-+

>--+---------------------------+-- )--------------------------><
   +-,--style----+-----------+-+
                 +-,--idstat-+
```

The AddInput method adds an entry line with a label (a static text) to the dialog.

Arguments:

The arguments are:

id

The unique ID of the entry line

attrName

The attrName is used to create an attribute in the dialog object that reflects the contents of the entry line (see AddEntryLine). If it is skipped, the text label is used as the attribute name.

x, y

The position of the upper-left edge of the label. The entry line is aligned automatically.

cx1

The length of the label. If omitted, the length is calculated.

cx2

The length of the entry field

cy

The height of the entry field. If omitted, the height is calculated.

text

The label displayed in front of the entry field

options

Possible options are:

HIDDEN

Makes the input group invisible

PASSWORD

Displays the password character instead of the typed-in characters

For further options see AddEntryLine.

idstat

An ID for the label

Example:

The following example creates an entry field and the label Your e-mail address (placed on the entry field's left side). It also creates an attribute with the name YOUREMAILADDRESS. The height of the elements is calculated. (See Sample Input Field.)

```
MyUserDialog~AddInput(402, , 20, 30, , 150, , "Your eMail address")
```

Figure 6-3. Sample Input Field

6.9.15. AddInputGroup

```
>>-aUserDialog~AddInputGroup(--startid--,--x--,--y--,--+----+--,-->
                                                       +-cx-+

          +------+
```

```
         V      |
>--,--"----text-+--"--+-----------------------------------+--)----><
                      +-,--+------------+--+-----------+-+
                           +-"NOBORDER"-+  +-,--idstat-+
```

The AddInputGroup method creates a group of one or more entry lines.

Arguments:

The arguments are:

startid

An ID that is assigned to the first entry line. Consecutive numbers are assigned to the other entry fields.

x, y

Position of the input group's upper-left corner

cx1

Length of the entry field label. If omitted, the length is calculated.

cx2

Length of the entry field in dialog units

text

The text strings used for each entry field's label. The single strings are to be separated by blank spaces. This argument determines the number of entry fields in total.

options

In addition to the options of AddInput, NOBORDER can be used to prevent the method from placing a group box around the group.

idstat

The ID of the first label. Usually you do not have to specify this value because labels are static controls.

Example:

The following example creates a four-line input group. The single entry lines are accessible by IDs 301 through 304.

```
MyDialog~AddInputGroup(301, 20, 20, ,130, "Name FirstName Street City")
```

Note: If you want to use labels that include blanks (for example, "First Name" instead of "FirstName"), use the AddInputStem method.

6.9.16. AddComboInput

```
>>-aUserDialog~AddComboInput(--id--,--+----------+--,--x--,--y--,-->
                                      +-attrName-+

>--+-----+--,--cx2--,--+--------+--,--text--,---------------------->
   +-cx1-+             +-clines-+

>--+----------------------------------------------+--)------------><
   +-,--+-----------------------+--+-----------+-+
        |    +-------------+    |  +-,--idstat-+
        |    V             |    |
        +-"----+-SIMPLE--+-+--"-+
               +-LIST----+
               +-SORT----+
               +-VSCROLL-+
```

The AddComboInput method adds a combo box and a label string to the dialog.

Arguments:

The arguments are:

id

The ID of the combo box

attrName

The name of the combo box. This name is used as an object attribute name.

x, y

Position of the group (text string of combo box)

cx1

Length of the text string

cx2

Width of the combo box

clines

Vertical length of the combo box in number of lines

text

Label being displayed on the left-hand side of the combo box

options

See AddComboBox

idstat

The ID of the label. Usually you do not have to specify this value because labels are static controls.

6.9.17. AddInputStem

```
>>-aUserDialog~AddInputStem(--startid--,--x--,--y--,--+-----+--,-->
                                                      +-cx1-+

>--cx2--,--textStem.--+-------------------------------+--)----->&lt;
                      +-,--+---------+--+-----------+-+
                           +-options-+  +-,--idstat-+
```

The AddInputStem method adds a group of input fields to the dialog. The difference between this method and the AddInputGroup method is that the titles (and names) of the single lines are passed to the method in a stem variable. Thus it is possible to use strings containing blank spaces.

Arguments:

The arguments are:

startid

The ID of the first entry line

x, y

The position of the whole group (upper-left corner)

cx1

The length of the text strings. If omitted, the size is calculated.

cx2

The width of the entry fields

textStem.

A stem variable containing all labels for the entry fields. The object attribute for each field is created on the basis of this string.

options

In addition to the options of the AddInput method, NOBORDER can be used to prevent the method from placing a group box around the group.

idstat

The ID of the first label. Usually you do not have to specify this value because labels are static controls.

Example:

The following example shows how to use AddInputStem. It creates a four-line input group. For each entry line (with IDs 401 through 404) an object attribute is provided. The names might be different from the title because not all characters can be used for Object Rexx symbols. In this example the NAME, FIRSTNAME, STREETNUMBER, and CITYZIP attributes are added to the object.

```
FNames.1="Name"
FNames.2="First Name"
FNames.3="Street & Number"
FNames.4="City & ZIP"

MyDialog~AddInputStem(401, 20, 20, , 150, FNames.)
```

6.9.18. AddCheckBoxStem

```
>>-aUserDialog~AddCheckBoxStem(--startid--,--x--,--y--,--------->

>--+----+--,--textStem.--,--max--+---------------+------------->
   +-cx-+                       +-,--+---------+-+
                                     +-options-+

>--+-------------------------------------------------------+-->
   +-+-----------------------------------------------------+-+
     +-,--+--------+--+---------------------------------+-+
          +-idstat-+  +-,--+----------+--+-------------+-+
                           +-fontName-+  +-,--fontSize-+

>--)---------------------------------------------------------><
```

The AddCheckBoxStem method creates a group of check box controls. Unlike the AddCheckGroup method you pass the titles of the check boxes in a stem variable instead of using a string. Thus you can use labels including blanks.

Arguments:

See AddCheckGroup for a description of the arguments. The new arguments are:

textStem.

A stem variable containing all labels for the check boxes. The object attribute for each check box is created on the basis of this string.

max

The maximum number of check box items in one column. If textStem has more items than max, a second column is created.

fontName

The name of the font used within the dialog

fontSize

The size of the font used within the dialog

Example:

The following example creates a three-column check box group:

```
CBNames.1="C"
CBNames.2="Pascal"
CBNames.3="Cobol"
CBNames.4="Rexx"
CBNames.5="Basic"
CBNames.6="Fortran"

MyDialog~AddCheckBoxStem(501, 20, 20, ,CBNames, 2, ,
"NOBORDER", 551, "Courier New", 12)
```

6.9.19. AddRadioStem

```
>>-aUserDialog~AddRadioStem(--startid--,--x--,--y--,--+----+--,--textStem.--,--max-->
                                                      +-cx-+

>--+-----------------------------------------------------------------------+-->
   +-,--+---------+--+-----------------------------------------------------+-+
        +-options-+  +-,--+--------+--+---------------------------------+-+
                           +-idstat-+  +-,--+----------+--+-------------+-+
                                            +-fontName-+  +-,--fontSize-+

>--)-------------------------------------------------------------><
```

The AddRadioStem method adds a group of radio button controls to the dialog.

See AddRadioGroup for a description of the arguments and an example.

6.9.20. AddScrollBar

```
>>-aUserDialog~AddScrollBar(--id--,--x--,--y--,--cx--,--cy------>

>--+-------------------------------+-- )------------------------><
   |       +-----------------+    |
   |       V                 |    |
   +-,--"----+-VERTICAL----+-+--"-+
             +-HORIZONTAL--+
             +-TOPLEFT-----+
             +-BOTTOMRIGHT-+
             +-TAB---------+
```

The AddScrollBar method adds a scroll bar to the dialog.

Arguments:

The arguments are:

id

This must be a unique number

x, y

The position of the upper-left corner relative to the dialog's client area measured in dialog units

cx

The horizontal size of the scroll bar in dialog units

cy

The vertical size of the scroll bar

options

The last argument can be one or more of:

VERTICAL

The scroll bar is positioned vertically (default)

HORIZONTAL

The scroll bar is positioned horizontally

TOPLEFT

The scroll bar is aligned to the top left of the given rectangle and has a predetermined width (if vertical) or height (if horizontal)

BOTTOMRIGHT

The scroll bar is aligned to the bottom right of the given rectangle and has a predetermined width (if vertical) or height (if horizontal)

TAB

The scroll bar is assigned a tab stop

6.9.21. AddButtonGroup

```
>>-aUserDialog~AddButtonGroup(--x--,--y--,--+----+--,--+----+--,--"-->
                                            +-cx-+     +-cy-+

   +---------------------+
   V                     |
>----text-- --id-- --msg-+--"---------------------------------------->
```

```
>--+-------------------------------+--)----------------------------><
   +-,--+--------+--+------------+-+
        +-bottom-+  +-,--options-+
```

Use the AddButtonGroup method to add more than one push button at once to the dialog. The buttons are arranged in a row or in a column.

Arguments:

The arguments are:

x, y

The position of the entire button group

cx, cy

The size of a single button. One or both arguments can be skipped. If so, the default values (cx=40, cy=12) are taken.

text ID msg

These arguments are interpreted as one string containing three words (separated by blanks) for each button. The first word is the text that is displayed on the button, the second is the ID of the button, and the third is the name of a message that is sent to the object whenever the button is clicked. The fourth to sixth words are for the next button, and so forth.

bottom

This is a flag to switch between a vertical (=0) or horizontal (=1) placement of the buttons.

options

If DEFAULT is used, the first button becomes the default button. For the other options, see AddButton.

Example:

The following example creates three buttons (Add, Delete, and Update):

```
MyDialog~AddButtonGroup(20, 235, , , "&Add 301 AddItem"  ,
"&Delete 302 DeleteItem"  ,
"&Update 303 UpdateItem")
```

6.10. Frames and Rectangles

The methods listed below add simple graphical elements to the dialog. They are useful for giving the dialog a nice finish. Use Frames and Rectangles in 3D Style to help you find the right element.

Figure 6-4. Frames and Rectangles in 3D Style

Note: There is currently no difference between rectangles and frames.

6.10.1. AddWhiteRect

```
>>-aUserDialog~AddWhiteRect(--x--,--y--,--cx--,--cy------------->

>--+-----------------------------------+--)------------------><
   |       +------------+               |
   |       V            |               |
   +-,--"----+-HIDDEN-+-+--"--+-------+-+
             +-BORDER-+       +-,--id-+
```

The AddWhiteRect method adds a white rectangle to the dialog.

Arguments:

The arguments are:

x, y

The position of the rectangle's upper-left corner relative to the dialog measured in dialog units

cx, cy

The size of the rectangle in dialog units

options

The options can be:

HIDDEN

The frame or rectangle is not visible at startup time

BORDER

A border is drawn around the rectangle or frame

id

The ID of the item, -1 is used by default

6.10.2. AddWhiteFrame

The AddWhiteFrame method is currently identical to the AddWhiteRect method.

6.10.3. AddGrayRect

```
>>-aUserDialog~AddGrayRect(--x--,--y--,--cx--,--cy-------------->

>--+----------------------------------------+--)--------------><
   +-,--+---------------------+--+-------+-+
        |    +------------+    |  +-,--id-+
        |    V            |    |
        +-"----+-HIDDEN-+-+--"-+
               +-BORDER-+
```

The AddGrayRect method adds a gray rectangle to the dialog.

Arguments:

See AddWhiteRect for a description of the arguments.

6.10.4. AddGreyFrame

The AddGreyFrame method is currently identical to the AddGreyRect method.

6.10.5. AddBlackRect

```
>>-aUserDialog~AddBlackRect(--x--,--y--,--cx--,--cy------------->

>--+----------------------------------------+--)--------------><
   +-,--+---------------------+--+-------+-+
        |    +------------+    |  +-,--id-+
        |    V            |    |
        +-"----+-HIDDEN-+-+--"-+
               +-BORDER-+
```

The AddBlackRect method adds a black rectangle to the dialog.

Arguments:

See AddWhiteRect for a description of the arguments.

6.10.6. AddBlackFrame

The AddBlackFrame method is currently identical to the AddBlackRect method.

6.11. OK and Cancel Push Buttons

The four methods described in this section add OK and Cancel push buttons to the dialog. The standard IDs (1 for OK and 2 for Cancel) are assigned to the buttons.

6.11.1. AddOkCancelRightBottom

```
>>-aUserDialog~AddOkCancelRightBottom-------------------------><
```

The AddOkCancelRightBottom method adds an OK and a Cancel push button to the lower-right edge of the dialog.

Example:

The following example adds the two push buttons to the bottom of the dialog. It further overrides the standard OK and Cancel methods.

```
::class MyClass subclass UserDialog

::method DefineDialog
      .
      .
      .
   self~AddOkCancelRightBottom

::method OK
   ret = MessageBox("Are you sure?", "Please confirm", "OK")
   if ret=1 then self~OK:super

::method Cancel
   ret = MessageBox("Do you really want to quit?", "Please confirm", "OK")
   if ret=1 then self~CANCEL:super
```

6.11.2. AddOkCancelLeftBottom

```
>>-aUserDialog~AddOkCancelLeftBottom--------------------------><
```

The AddOkCancelLeftBottom method adds an OK and a Cancel push button to the lower-left edge of the dialog.

6.11.3. AddOkCancelRightTop

```
>>-aUserDialog~AddOkCancelRightTop-----------------------------><
```

The AddOkCancelRightTop method adds an OK and a Cancel push button vertically to the upper-right edge of the dialog.

6.11.4. AddOkCancelLeftTop

```
>>-aUserDialog~AddOkCancelLeftTop------------------------------><
```

The AddOkCancelLeftTop method adds an OK and a Cancel push button vertically to the upper-left edge of the dialog.

6.12. Dialog Control Methods

The methods described in this section control the execution of the dialog; they are for internal use only.

6.12.1. StartIt

```
>>-aUserDialog~StartIt(--+------+--+-----------+--)-------------><
                        +-icon-+  +-,--modal--+
```

The StartIt method is intended for internal use only. It is used to create a real Windows object based on a dialog template. For a `RcDialog` the template is supplied by a resource script file. For a `UserDialog` the template is created by using the Add... methods. The method is called by the Execute or ExecuteAsync methods.

Internal Use:

This method is intended for internal use. It can be overridden, although that is not recommended.

Arguments:

There are two optional arguments:

icon

The resource ID of the icon to be used as the dialog icon. When this argument is omitted the default dialog icon is used.

modal

If a modal or a modeless dialog is to be created. A value of 0 signals a modal dialog should be created. A value of 1, or the string "NOTMODAL" case insignificant, signals a modeless dialog should be created. The default is 0.

6.12.2. StopIt

```
>>-aUserDialog~StopIt----------------------------------------><
```

The StopIt method is for internal use only. It is the counterpart to the BaseDialog class StopIt method to remove the Windows object.

Protected:

This method is protected and cannot be called from outside the instance. It can be overwritten, although this is not recommended.

6.13. Menu Methods

The methods described in this section are for creating dialog menus.

6.13.1. CreateMenu

```
>>-aUserDialog~CreateMenu(--+-------+--)----------------------><
                           +-count-+
```

This method initializes the creation of a menu. When you have finished adding menu items, call SetMenu to combine the menu with the dialog.

Arguments:

There is only one argument:

count

Maximum number of menu items that can be added

6.13.2. AddPopupMenu

```
>>-aUserDialog~AddPopupMenu(--name--,--"--options--"--)-------><
```

This method adds a popup menu to the menu.

The last popup menu must have the option "END" specified.

Arguments:

The arguments are:

name

The name of the popup menu

options

GRAYED DISABLED END

6.13.3. AddMenuItem

```
>>-aUserDialog~AddMenuItem(--name--,--id--,--"--options--"--,--msgToRaise--)-><
```

This method adds a new menu item after the last added item.

The last menu item in a popup menu must have the option "END" specified.

Arguments:

The arguments are:

name

The name of the menu item. The name can include &.

id

The ID of the menu item

options

GRAYED DISABLED END CHECKED

msgToRaise

Method to be called when the menu item is selected. See ConnectMenuItem.

6.13.4. AddMenuSeparator

```
>>-aUserDialog~AddMenuSeparator-------------------------------><
```

This method adds a menu separator to the menu after the last added item.

6.13.5. SetMenu

```
>>-aUserDialog~SetMenu----------------------------------------><
```

This method adds the menu that was created or loaded for the current dialog to the dialog window. Note that the menu needs additional space and therefore displaces the rest of the dialog items.

6.13.6. LoadMenu

```
>>-aUserDialog~LoadMenu(--resfile--,--menuid-------------------->

>--+-------------------------------------+--)------------------><
   +-,--+----------------+--+----------+-+
        +-"CONNECTITEMS"-+  +-,--count-+
```

This method loads a menu resource out of a resource script.

To combine the menu with the dialog, you must call SetMenu.

Arguments:

The arguments are:

resfile

The name of a resource script.

menuid

The ID of the menu resource.

CONNECTITEMS

The menu items are automatically connected to methods named after the name of the menu item. See LoadItems for further details.

count

The maximum number of menu items that can be loaded.

Chapter 7. PlainUserDialog Class and PlainBaseDialog Class

The PlainUserDialog class subclasses from PlainBaseDialog class and provides all the methods that normally are required to execute a dialog that is either created dynamically or loaded from a resource script (.RC). In other words it is a limited version of UserDialog Class. Use `::requires "OODPLAIN.CLS"` in your script to get access to the PlainUserDialog class.

> **Note:** If you use `::requires "OODWIN32.CLS"` you do not need to specify the `::requires "OODIALOG.CLS"` directive because OODIALOG.CLS is included from OODWIN32.CLS.

> **Note:** If you use `::requires "OODIALOG.CLS"` you do not need to specify the `::requires "OODPLAIN.CLS"` directive because OODPLAIN.CLS is included from OODIALOG.CLS.

UserDialog includes all the methods PlainUserDialog does plus all the methods defined in the DialogExtensions class. These are more specific methods that cover asynchronous dialog execution, scroll bar support, resizing and repositioning, bitmaps, graphics (device context related methods), scrolling text, and menus (action bars).

The reason for splitting the functionality into two classes is to provide a smaller package which requires less system resources for ordinary user interfaces like the standard dialogs.

The following table lists all the methods that are provided by the PlainUserDialog class. The individual methods are documented in BaseDialog Class or UserDialog Class.

Attributes:

Instances of the PlainUserDialog class have the following attributes:

AutoDetect

Automatic data field detection on (=1, default) or off (=0). For the UserDialog subclass the default is off and Connect... methods or a resource script are usually used.

AutomaticMethods

A queue containing the methods that are started concurrently before the execution of the dialog

ConstDir

A directory string storing the numerical values assigned to symbolic IDs (#define-statements in the resource script)

DataConnection

Protected attribute to store connections between dialog items and the attributes of the dialog instance

DlgHandle

A handle to the dialog

Finished

0 if dialog is executing, 1 if terminated with OK, and 2 if canceled

InitCode

After the Init method has finished executing the value of the attribtute is 0 if no errors were detected during initialization, otherwise its value is non-zero. After instantiating a new dialog object, the InitCode attribute should be checked. If its value is not 0 then there was some problem initializing the object. The programmer should treat this as an error condition and not expect that the underlying Windows dialog can be created successfully.

After the dialog is finished, the attribute will be 1 if the user terminated the dialog with the Ok button. Its value will be 2 if the user canceled the dialog.

UseStem

Protected attribute that is true (=1) if a stem variable was passed to init

Methods:

Instances of the PlainUserDialog class implement the methods listed in PlainUserDialog and PlainBaseDialog Class Methods table.

Table 7-1. PlainUserDialog and PlainBaseDialog Class Methods

Method...	...on page
AddAttribute	AddAttribute
AddBitmapButton	AddBitmapButton
AddBlackFrame	AddBlackFrame
AddBlackRect	AddBlackRect
AddButton	AddButton
AddButtonGroup	AddButtonGroup
AddCheckBox	AddCheckBox
AddCheckBoxStem	AddCheckBoxStem
AddCheckGroup	AddCheckGroup
AddComboBox	AddComboBox
AddComboEntry	AddComboEntry
AddComboInput	AddComboInput
AddEntryLine	AddEntryLine
AddGrayFrame	AddGreyFrame
AddGrayRect	AddGrayRect
AddGroupBox	AddGroupBox
AddIcon	AddIcon

Method...	...on page
AddInput	AddInput
AddInputGroup	AddInputGroup
AddInputStem	AddInputStem
AddListBox	AddListBox
AddListEntry	AddListEntry
AddMenuItem	AddMenuItem
AddMenuSeparator	AddMenuSeparator
AddOkCancelLeftBottom	AddOkCancelLeftBottom
AddOkCancelLeftTop	AddOkCancelLeftTop
AddOkCancelRightBottom	AddOkCancelRightBottom
AddOkCancelRightTop	AddOkCancelRightTop
AddPasswordLine	AddPasswordLine
AddPopupMenu	AddPopupMenu
AddRadioButton	AddRadioButton
AddRadioGroup	AddRadioGroup
AddRadioStem	AddRadioStem
AddScrollBar	AddScrollBar
AddText	AddText
AddUserMsg	AddUserMsg
AddWhiteFrame	AddWhiteFrame
AddWhiteRect	AddWhiteRect
AssignWindow	AssignWindow
AutoDetection	AutoDetection
Cancel	Cancel
Center	Center
ChangeComboEntry	ChangeComboEntry
ChangeListEntry	ChangeListEntry
ClearMessages	ClearMessages
ComboAddDirectory	ComboAddDirectory
ComboDrop	ComboDrop
ConnectButton	ConnectButton
ConnectCheckBox	ConnectCheckBox
ConnectComboBox	ConnectComboBox
ConnectControl	ConnectControl
ConnectEntryLine	ConnectEntryLine
ConnectList	ConnectList
ConnectListBox	ConnectListBox
ConnectListLeftDoubleClick	ConnectListLeftDoubleClick

Method...	...on page
ConnectMultiListBox	ConnectMultiListBox
ConnectRadioButton	ConnectRadioButton
Create	Create
CreateCenter	CreateCenter
CreateMenu	CreateMenu
DefineDialog	DefineDialog
DeInstall	DeInstall
DeleteComboEntry	DeleteComboEntry
DeleteListEntry	DeleteListEntry
Disable	Disable
DisableItem	DisableItem
Enable	Enable
EnableItem	EnableItem
Execute	Execute
FindComboEntry	FindComboEntry
FindListEntry	FindListEntry
FocusItem	FocusItem
Get	Get
GetAttrib	GetAttrib
GetButtonRect	GetButtonRect
GetCheckBox	GetCheckBox
GetComboEntry	GetComboEntry
GetComboItems	GetComboItems
GetComboLine	GetComboLine
GetCurrentComboIndex	GetCurrentComboIndex
GetCurrentListIndex	GetCurrentListIndex
GetData	GetData
GetDataStem	GetDataStem
GetEntryLine	GetEntryLine
GetID	GetID
GetItem	GetItem
GetListEntry	GetListEntry
GetListItems	GetListItems
GetListLine	GetListLine
GetMultiList	GetMultiList
GetPos	GetPos
GetRadioButton	GetRadioButton
GetSize	GetSize

Method...	...on page
GetTextSize	GetTextSize
GetValue	GetValue
HandleMessages	HandleMessages
Help	Help
Hide	Hide
HideItem	HideItem
HideWindow	HideWindow
Init	Init
InitAutoDetection	InitAutoDetection
InitDialog	InitDialog
InsertComboEntry	InsertComboEntry
InsertListEntry	InsertListEntry
IsDialogActive	IsDialogActive
ItemTitle	ItemTitle
Leaving	Leaving
ListAddDirectory	ListAddDirectory
ListDrop	ListDrop
Load	Load
LoadFrame	LoadFrame
LoadItems	LoadItems
LoadMenu	LoadMenu
Move	Move
NoAutoDetection	NoAutoDetection
OK	OK
Resize	Resize
Run	Run
SetAttrib	SetAttrib
SetCheckBox	SetCheckBox
SetComboLine	SetComboLine
SetCurrentComboIndex	SetCurrentComboIndex
SetCurrentListIndex	SetCurrentListIndex
SetData	SetData
SetDataStem	SetDataStem
SetEntryLine	SetEntryLine
SetListLine	SetListLine
SetListTabulators	SetListTabulators
SetMenu	SetMenu
SetMultiList	SetMultiList

Method...	...on page
SetRadioButton	SetRadioButton
SetStaticText	SetStaticText
SetTitle	SetTitle
SetValue	SetValue
SetWindowTitle	SetWindowTitle
Show	Show
ShowItem	ShowItem
ShowWindow	ShowWindow
StartIt	StartIt
StopIt	StopIt
Title	Title
Title=	Title=
Update	Update
Validate	Validate

Chapter 8. Resource File Dialogs

The resource definition for a dialog can either be constructed manually, (see the UserDialog Class,) or the definition can be read from a file. ooDialog provides two main dialog classes that read their dialog definitions from a file: the ResDialog class and the RcDialog class.

The `ResDialog` class uses a dialog definition from a binary file and a `RcDialog` class gets the definition from a resource script file. Resource script files are plain text files. In general, both types of files are produced by resource editors which allow the user to visually design the dialog in a "What You See Is What You Get" manner. Since resource definitions are standardized, resource editors that produce Windows compatible files can be used to define dialogs for use with ooDialog.

8.1. ResDialog Class

The ResDialog class is designed to be used together with a binary (compiled) resource. A binary dialog resource is linked to a DLL (that is, a file with the extension .DLL).

Requires:

`ResDlg.cls` is the source file of this class.

```
::requires "oodialog.cls"
```

Subclass:

The ResDialog class is a subclass of BaseDialog and therefore has all the methods defined for that class.

8.1.1. Init

```
>>-aResDialog~Init(--Library--,--Resource--+-------------+--+----------+--)-->&lt;
                                           +-,--DlgData.-+  +-,--hFile-+
```

The Init method of the parent class, `BaseDialog`, has been overridden.

Arguments:

The arguments when creating a new dialog instance of this class are:

Library

The file name of the DLL where the resource is located.

Resource

The ID of the resource. This is an unique number, or sybmbolic ID, you assigned to the (dialog) resource while creating it.

DlgData.

A stem variable (don't forget the trailing period) that contains initialization data.

hFile

The file name of a header file containing symbolic ID defines for resources.

Example:

This sample code creates a new dialog object from the ResDialog class. It uses dialog resource 100 in the `MYDLG.DLL` file. The dialog is initialized with the values of the MyDlgData. stem variable.

```
MyDlgData.101="1"
MyDlgData.102="Please enter your password."
MyDlgData.103=""

dlg = ResDialog~new("MYDLG.DLL", 100, MyDlgData.)
if dlg~initCode <> 0 then do
  say 'The dialog could not be created'
  return .false
  /* Or some other error handling code. */
end
  ...
```

An additional example can be found under the `Init` method, in the BaseDialog section, that shows the use of the dialog data stem and header file arguments in more detail.

Note: For a `ResDialog` the `InitCode` attribute will be -1 after the dialog object is instantiated if the resource ID could not be resolved. For other cases where initialization failed the value will be some other non-zero value.

8.1.2. StartIt

```
>>-aResDialog~StartIt(--+------+--+-----------+--)-------------><
                        +-icon-+  +-,--modal--+
```

The StartIt method is intended for internal use only. It is used to create the real Windows object based on the dialog template in the binary resource file. It is called by the Execute or ExecuteAsync methods.

Internal Use:

This method is intended for internal use. It can be overridden, although that is not recommended.

Arguments:

There are two optional arguments:

icon

The resource ID of the icon to be used as the dialog icon. When this argument is omitted the default dialog icon is used.

modal

If a modal or a modeless dialog is to be created. A value of 0 signals a modal dialog should be created. A value of 1, or the string "NOTMODAL" case insignificant, signals a modeless dialog should be created. The default is 0.

8.1.3. SetMenu

```
>>-aResDialog~SetMenu(--resid--)------------------------------><
```

The SetMenu method adds a menu resource, that is stored in the same DLL, to the dialog. Note that the menu needs additional space and therefore displaces the rest of the dialog items.

SetMenu can be called in the InitDialog method only.

Arguments:

There is only one argument:

resid

ID of the menu resource stored in the same DLL as the dialog.

8.2. RcDialog Class

The `RcDialog` class gets its dialog definition from a resource script file. Resource script files are plain text files usually produced by a resource editor. The files generally have a file extension of ".rc", but an extension of ".dlg" is used by some resource editors.

This class subclasses the UserDialog class and has available all the methods of that class. `RcDialog` is a convenience class that uses the Load method of the `UserDialog` to load the dialog definition from the resource script file. The programmer then does not need to worry about the details of how and when to load the resource script. There is a slight difference in instantiating the class, and the programmer does not need to use the DefineDialog method because the dialog is defined in the resource script. Other than that, the `RcDialog` class is used in the same manner as the `UserDialog` class.

Note: Although the programmer does not need to use the `Define` method, there is nothing preventing its use. The `Define` method could be used at runtime to add additional dialog elements to those defined in the resource script.

Requires:

`RcDialog.cls` is the source file for this class.

```
::requires "oodialog.cls"
```

Subclass:

The `RcDialog` class is a subclass of `UserDialog`.

8.2.1. Init

```
>>-aRcDialog~Init(--script--,--dlgID--+-------------+--+----------+--------->
                                      +-,--dlgData.-+  +-,--hFile-+

>--+-------------------------------------------------------+--)------------><
   +-,--+-----------------------------+--+-------------+-+
        |    .--------------------.    |  +-,--expected-+
        |    V                    |    |
        +-"----+-CENTER---------+-+--"-+
               +-CONNECTBUTTONS-+
               +-CONNECTRADIOS--+
               +-CONNECTCHECKS--+
```

The Init method of the parent class, `UserDialog`, has been overridden.

Arguments:

The arguments when creating a new dialog instance of this class are:

script

The name of the resource script file where the dialog is defined.

dlgID

The resource ID of the dialog. This is the unique number, or sybmbolic ID, you assigned to the (dialog) resource when creating it.

dlgData.

A stem variable (don't forget the trailing period) that contains initialization data.

hFile

The file name of a header file containing the symbolic ID defines for the resources used by the program.

options

One or more of the keywords listed in the syntax diagram, separated by blanks:

CENTER

The dialog is to be positioned in the center of the screen.

CONNECTBUTTONS

For each push button a connection to an object method in the `RcDialog` object is established automatically. See ConnectButton for a description of connecting buttons to a method. The name for the object method is created by ooDialog and the programmer needs to define the object method. The method name will be the button text with all spaces, ampersands, and colons removed.

CONNECTRADIOS

Similar to CONNECTBUTTONS, but this option will connect the radio buttons to an object method. For radio buttons, the created method name will be the button text with all spaces, ampersands, and colons removed, prefaced with "ID".

CONNECTCHECKS

Exactly the same as CONNECTRADIOS, for check box controls. The object method name is generated in the same way as it is for radio buttons.

expected

This is the maximum number of dialog elements the dialog object can handle. See the Create method for details. The default value for this argument is 200.

Example:

This example creates a new dialog object using the `SimpleLB` class, which subclasses the `RcDialog`. It uses the dialog defintion with symbolic ID `IDD_DIALOG1` in the `listBox.rc` resource script file. The dialog is initialized so that the "Doctor" item is selected in the list box with symbolic ID of `IDC_LB` by using the `dlgData.` stem variable. The symbolic IDs for this program are contained in the "resource.h" file. When the user closes the dialog, the `dlgData.IDC_LB` variable will contain the text of the selected item in the list box.

```
/* Simple ListBox using a .rc file for the dialog definition */

  dlgData.IDC_LB = "Doctor"

  dlg = .SimpleLB~new("listBox.rc", IDD_DIALOG1, dlgData., "resource.h")
  if dlg~initCode = 0 then do
    dlg~Execute("SHOWTOP")
    dlg~Deinstall
  end
  else do
    say "Problem creating the dialog.  Init code:" dlg~initCode
    return 99
  end
```

```
  say "The user's choosen profession is" dlgData.IDC_LB
  return 0
-- End of entry point.

::requires "OODWIN32.CLS"

::class SimpleLB subclass RcDialog inherit AdvancedControls

::method initDialog

  lb = self~getListBox(IDC_LB)

  lb~add("Business Manager" )
  lb~add("Software Developer")
  lb~add("Accountant")
  lb~add("Lawyer")
  lb~add("Doctor")
```

To try the example, cut and paste the above code into a file named `listBox.rex` and paste the following code into files `listBox.rc` and `resource.h`.

```
/* listBox.rc */
#include <windows.h>
#include <commctrl.h>
#include "resource.h"

IDD_DIALOG1 DIALOGEX 30,30,179,182
STYLE DS_MODALFRAME | DS_FIXEDSYS | WS_VISIBLE | WS_CAPTION | WS_POPUP | WS_SYSMENU
CAPTION "List Box Example"
FONT 8,"MS Shell Dlg 2",400,0,1
BEGIN
    DEFPUSHBUTTON   "Close",IDOK,122,161,50,14
    LISTBOX         IDC_LB,8,28,163,122,WS_TABSTOP | LBS_NOINTEGRALHEIGHT | LBS_SORT
    CTEXT           "Pick Your Choosen Profession",IDC_STATIC,26,13,122,9
END

/* resource.h */
#ifndef IDC_STATIC
#define IDC_STATIC (-1)
#endif

#define IDD_DIALOG1                     100
#define IDC_LB                          1001
```

Chapter 9. CategoryDialog Class

The CategoryDialog class creates and controls a dialog that has more than one panel. It is similar to the notebook control available in OS/2 or the property sheet available in the Windows 95 user interface.

Depending on the style you choose, you can switch among different pages by either clicking radio buttons or selecting an item from a drop down list. Each page has its own window controls.

Requires:

`CatDlg.cls` is the source file of this class.

```
::requires "oodialog.cls"
```

Subclass:

The CategoryDialog class is a subclass of UserDialog (see UserDialog Class).

Attributes:

Instances of the CategoryDialog class have the following attributes:

Catalog

A directory describing the layout and behavior of the dialog. This directory is usually set up in the InitCategories method of the dialog (see InitCategories for more information).

StaticID

An internal counter

Methods:

Instances of the CategoryDialog class implement the methods listed in the following table.

Note: In fact, most of the methods do the same as the methods in the parent class, UserDialog, except that they are enabled to work with a category dialog.

Method...	...on page
AddCategoryComboEntry	AddCategoryComboEntry
AddCategoryListEntry	AddCategoryListEntry
CategoryComboAddDirectory	CategoryComboAddDirectory
CategoryComboDrop	CategoryComboDrop
CategoryListAddDirectory	CategoryListAddDirectory
CategoryListDrop	CategoryListDrop
CategoryPage	CategoryPage
ChangeCategoryComboEntry	ChangeCategoryComboEntry
ChangeCategoryListEntry	ChangeCategoryListEntry

Method...	...on page
ChangePage	ChangePage
CreateCategoryDialog	CreateCategoryDialog
CurrentCategory	CurrentCategory
DefineDialog	DefineDialog
DeleteCategoryComboEntry	DeleteCategoryComboEntry
DeleteCategoryListEntry	DeleteCategoryListEntry
DisableCategoryItem	DisableCategoryItem
EnableCategoryItem	EnableCategoryItem
FindCategoryComboEntry	FindCategoryComboEntry
FindCategoryListEntry	FindCategoryListEntry
FocusCategoryItem	FocusCategoryItem
GetCategoryAttrib	GetCategoryAttrib
GetCategoryCheckBox	GetCategoryCheckBox
GetCategoryComboEntry	GetCategoryComboEntry
GetCategoryComboItems	GetCategoryComboItems
GetCategoryComboLine	GetCategoryComboLine
GetCategoryEntryLine	GetCategoryEntryLine
GetCategoryListEntry	GetCategoryListEntry
GetCategoryListItems	GetCategoryListItems
GetCategoryListLine	GetCategoryListLine
GetCategoryMultiList	GetCategoryMultiList
GetCategoryRadioButton	GetCategoryRadioButton
GetCategoryValue	GetCategoryValue
GetCurrentCategoryComboIndex	GetCurrentCategoryComboIndex
GetCurrentCategoryListIndex	GetCurrentCategoryListIndex
GetSelectedPage	GetSelectedPage
HideCategoryItem	HideCategoryItem
Init	Init
InitCategories	InitCategories
InitDialog	InitDialog
InsertCategoryComboEntry	InsertCategoryComboEntry
InsertCategoryListEntry	InsertCategoryListEntry
MoveCategoryItem	MoveCategoryItem
NextPage	NextPage
PageHasChanged	PageHasChanged
PreviousPage	PreviousPage
ResizeCategoryItem	ResizeCategoryItem
SendMessageToCategoryItem	SendMessageToCategoryItem
SetCategoryAttrib	SetCategoryAttrib

Method...	...on page
SetCategoryCheckBox	SetCategoryCheckBox
SetCategoryComboLine	SetCategoryComboLine
SetCategoryEntryLine	SetCategoryEntryLine
SetCategoryItemFont	SetCategoryItemFont
SetCategoryListLine	SetCategoryListLine
SetCategoryListTabulators	SetCategoryListTabulators
SetCategoryMultiList	SetCategoryMultiList
SetCategoryRadioButton	SetCategoryRadioButton
SetCategoryStaticText	SetCategoryStaticText
SetCategoryValue	SetCategoryValue
SetCurrentCategoryComboIndex	SetCurrentCategoryComboIndex
SetCurrentCategoryListIndex	SetCurrentCategoryListIndex
ShowCategoryItem	ShowCategoryItem
StartIt	StartIt

9.1. Setting Up the Dialog

The following methods are used to set up the pages of the dialog and start it.

9.1.1. Init

```
>>-aCategoryDialog~Init(--+----------+--,--+------+--,--+------+--,--+-------+--,-->
                          +-DlgData.-+     +-catx-+     +-caty-+     +-catcx-+

>--+--------------------+--,--+----------+--,--+----------+--,--+--------+--,------>
   +-"--+-DROPDOWN-+--"-+     +-cattable-+     +-catlabel-+     +-catmax-+
        +-TOPLINE--+
        +-WIZARD---+

>--+-------+--)--->&lt;
   +-hFile-+
```

The Init method initializes the category dialog object.

Arguments:

The arguments are:

DlgData.

A stem variable (do not forget the trailing period) that contains initialization data for some or all dialog items. If the dialog is terminated by means of the OK button, the values of the

dialog's data fields are copied to this variable. The ID of the dialog items is used to name the entry within the stem.

catx, caty

The position of the category selection control group (radio buttons or combo box). The defaults are 10 and 4.

catcx

This argument sets the length of one item of the control group (calculated if omitted)

style

This argument determines the style of the category dialog. Without one of the following keywords, the category selection is done by a vertical radio button group:

DROPDOWN

Creates a drop-down list at the top (useful if there are many categories)

TOPLINE

Draws a horizontal radio button group at the top of the client area

WIZARD

Adds Backward and Forward buttons with IDs 11 and 12 to switch between category pages

Without DROPDOWN and TOPLINE the default category selection is done by a vertical radio button group, with the dialog pages to the right of the radio buttons.

cattable

This argument can be used to set the category names separated by blanks. If omitted, set the category names in the InitCategories method.

catlabel

This argument defines the label for the combo box in DROPDOWN style (default is "Page:")

catmax

This argument sets the split point of the radio button group in default style, or the number of entries in the combo box drop-down list.

hFile

The file name of a header file containing symbolic ID defines for resources.

Example:

The following example creates a category dialog, using a combo box as the selection control:

```
dlg = MyCategoryDialog~new(MyData., , , ,"DROPDOWN", ,
"Movies Cinemas Days Ticket","Dialog panel:")
dlg~createCenter(200,180,"Let's go to the movies")
```

```
dlg~execute("SHOWTOP")
...
::class MyCategoryDialog subclass CategoryDialog

::method Movies        /* define the Movies page  */
...
::method Cinemas       /* define the Cinemas page */
...
```

An additional example can be found under the `Init` method, in the BaseDialog section. This example shows the use of the dialog data stem and header file arguments in more detail.

9.1.2. InitCategories

```
>>-aCategoryDialog~InitCategories------------------------------><
```

The InitCategories method is called by Init to set the characteristics of the category dialog.

Protected:

This method is protected.

Catalog directory:

The InitCategories should set up the Catalog directory with information about the layout and the behavior of the dialog. The directory entries are:

names

An array containing the names of the categories. The array is initialized with the names given in the Init method (argument cattable). These names are used as labels for page selection control and as messages sent to the object to define the single pages.

Your subclass must provide a method for each category page--with the same name as the label in this directory--to define the dialog page using LoadItems to load the dialog items from a resource script or Add... methods. Notice that blanks are removed when you call the Define... methods.

If your subclass provides methods with the prefix Init followed by the name of the categories (blanks removed), these methods are called by InitDialog to initialize the dialog (each page is a dialog) that contains the corresponding category.

Unless you already specified the categories with the Init method, you must assign an array to this Catalog entry.

count

Number of categories

handles

For internal use only

id

For internal use only

category

For internal use only

page

A directory with the following entries:

font

Name of the font used for the dialog

fsize

Size of the font

style

Style of the dialog (see Create)

expected

Total number of expected dialog items of all category pages (200)

btnwidth

Width of Backward and Forward push buttons (see WIZARD in Init method)

leftbtntext

Alternate label of Backward button

rightbtntext

Alternate label of Forward button

The next four entries should not be modified:

x

Horizontal position of the category pages relative to the parent dialog

y

Vertical position of the category pages relative to the parent dialog

w

Width of the category pages

h

Height of the category pages

Example:

The following example sets the category names to Text Editor, Compiler, Linker, and Debugger. The subclass of CategoryDialog must define four methods named after them.

```
::class MyCategoryDialog subclass CategoryDialog
::method InitCategories
   self~catalog["names"]= .array~of("Text Editor","Compiler","Linker", ,
   "Debugger")
   self~catalog["page"]["leftbtntext"] = "&Previous"
   self~catalog["page"]["rightbtntext"] = "&Next"

::method TextEditor       /* blanks are removed when the message is sent
                             from DefineDialog */
                          /* add control to the first page */
      .
      .
      .
::method Compiler
      .
      .
      .
::method InitTextEditor   /* called by InitDialog */
                          /* initialize the first dialog */
      .
      .
      .
```

9.1.3. DefineDialog

```
>>-aCategoryDialog~DefineDialog--------------------------------><
```

The DefineDialog method is called after the main dialog has been created. This method must not be overwritten in a subclass because it defines the layout of the frame window and calls the definition methods for each category page.

Protected:

This method is protected.

Example:

Assume that you have the categories "Common Data", "Company Data", and "Special". The following methods are automatically called by DefineDialog to add dialog items to the associated page:

CommonData

to add controls to the first category page.

CompanyData

to add controls to the second category page.

Special

to add controls to the third category page.

See InitCategories for more information.

9.1.4. CategoryPage

```
>>-aCategoryDialog~CategoryPage-------------------------------><
```

The CategoryPage method adds controls to the base window of a Category Dialog. It is used to define the layout of the parent dialog that contains the single pages.

Protected:

This method is protected and should not be overwritten or called. Use the InitCategories method to set up the dialog.

9.1.5. CreateCategoryDialog

```
>>-aCategoryDialog~CreateCategoryDialog(--x--,--y--,--cx--,--cy--,-->

>--+----------+--,--+----------+--,--options--,--expected--)---><
   +-fontName-+     +-fontSize-+
```

The CreateCategoryDialog method creates the category dialog.

Protected:

This method is protected. It is called by another method and usually does not have to be called manually.

9.1.6. InitDialog

```
>>-aCategoryDialog~InitDialog----------------------------------><
```

The InitDialog method is called after the Windows dialog has been created and before the category dialog is to be displayed.

Do not override this method to initialize your category dialog pages but define an Init... method for each of your pages that you want to initialize like adding combo and list box items. If your subclassed dialog defines a method that has the prefix "Init" followed by the name of the category (without blanks), this

method is called by InitDialog to handle the initialization of the corresponding page. If you use a method that requires a category specifier, such as AddCategoryComboEntry or GetTreeControl, and you omit the category, OODialog assumes that the dialog item addressed is part of the page that contains the current category.

Protected:

This method is protected.

Example:

Assume that you have the categories "Common Data", "Company Data", and "Special". The following methods are automatically called when provided by the subclass:

InitCommonData

to initialize the first category.

InitCompanyData

to initialize the second category.

InitSpecial

to initialize the third category.

See InitCategories for more information.

9.1.7. GetSelectedPage

```
>>-aCategoryDialog~GetSelectedPage----------------------------><
```

The GetSelectedPage method is used internally to return the currently selected page using the combo box or radio buttons (1 indicates the first page).

9.1.8. CurrentCategory

```
>>-aCategoryDialog~CurrentCategory----------------------------><
```

The CurrentCategory method returns the number of the current dialog page. The first page is numbered 1.

Example:

The following example tests the current page number:

```
if MyCategoryDialog~CurrentCategory=2 then do ...
```

9.1.9. NextPage

```
>>-aCategoryDialog~NextPage----------------------------------><
```

The NextPage method switches the dialog to the next category page.

9.1.10. PreviousPage

```
>>-aCategoryDialog~PreviousPage------------------------------><
```

The PreviousPage method switches the dialog to the previous category page.

9.1.11. ChangePage

```
>>-aCategoryDialog~ChangePage(--+---------+--)-----------------><
                               +-NewPage-+
```

The ChangePage method switches the dialog to another page and returns the new page number. It is also called by selection control to activate the selected page. ChangePage invokes PageHasChanged after the new page is activated.

Arguments:

The only argument is:

NewPage

The page number of the new page (default is the page selected by the combo box or radio button)

Example:

The following example activates the second category page:

```
MyCategoryDialog~ChangePage(2)
```

9.1.12. PageHasChanged

```
>>-aCategoryDialog~PageHasChanged(--oldpage--,--newpage--)-----><
```

The PageHasChanged method is invoked by ChangePage when a new page is activated. The default implementation returns without an action. The user can override this method to react to page changes.

Arguments:

The arguments are:

oldpage

The page number of the previous page

newpage

The page number of the new page

9.1.13. StartIt

```
>>-aCategoryDialog~StartIt------------------------------------><
```

The `StartIt` method is called by the Execute or ExecuteAsync methods to create a real Windows object based on a dialog template. In the `CategoryDialog` class the Windows object(s) are created in the `DefineDialog` method. Therefore, the `StartIt` method is overridden in this class to change the default behavior of the superclass. (If unchanged the superclass would create a second duplicate Windows object from the dialog template.)

You might override this method in your subclass, but be sure to forward the message to the parent method and to return the result to the caller.

Example:

This is a template for overridding this method:

```
::class MyCatDlg subclass CategoryDialog
::method StartIt
   say "this is method "StartIt" !"
   self~StartIt:super()
   handle = result
   ...
   return handle
```

9.2. Connect... Methods

```
>>-aCategoryDialog~Connect...(--id--,--fname--)----------------><
```

The Connect... methods connect data dialog items of certain types with the dialog object. The Connect... methods should be placed into the user-defined methods with the names of the categories defined in the InitCategories method. The Connect... methods are defined for the BaseDialog class. For more information, see BaseDialog Class.

Arguments:

The arguments are:

id

The ID of the dialog item

fname

The name of the object attribute

Example:

The following example connects an entry line in the Movies page with the FIRSTNAME object attribute:

```
::method InitCategories
   self~catalog["names"] = .array~of("Movies",...)
...
::method Movies
  self~ConnectEntryLine(101, "FIRSTNAME")
```

Note: IDs for dialog elements need not be unique across all pages. However, IDs for buttons and list boxes that are connected to methods must be unique for the whole category dialog.

9.3. Methods for Dialog Items

The methods listed in this section deal with individual dialog items on one of the pages of the category dialog.

The methods correspond to methods with similar names of the BaseDialog class; the word Category is inserted between the verb and the dialog item in the method name. For example, AddCategoryComboEntry for the CategoryDialog class has the same function as AddComboEntry of the BaseDialog class.

Note: The methods listed here have the same parameters as the corresponding methods of the BaseDialog class, with the number of the category page as an extra parameter.

Another way to directly address dialog items of a category dialog is to retrieve an object of the DialogControl class (see page DialogControl Class) or one of its derivates that is associated with the requested dialog control. To retrieve such an object, you can call one of the following methods depending on the requested control:

- GetStaticControl
- GetEditControl
- GetButtonControl

- GetRadioControl
- GetCheckControl
- GetListBox
- GetComboBox
- GetScrollBar
- GetTreeControl
- GetListControl
- GetProgressBar
- GetSliderControl
- GetTabControl

To use these methods and the resulting objects, your category dialog must inherit from the mixin class AdvancedControls. You can do this by adding the keyword "inherit" followed by the mixin class name "AdvancedControls". For example:

```
::class MyCategory subclass CategoryDialog public inherit AdvancedControls
```

9.4. Get and Set Methods

The following sections describe the individual Get and Set methods.

9.4.1. SetCategoryStaticText

```
>>-aCategoryDialog~SetCategoryStaticText(--id--,--data--,--category-->

>--)-----------------------------------------------------------><
```

For more information, see SetStaticText.

9.4.2. GetCategoryEntryLine

```
>>-aCategoryDialog~GetCategoryEntryLine(--id--,--category--)--->><
```

For more information, see GetEntryLine.

9.4.3. SetCategoryEntryLine

```
>>-aCategoryDialog~SetCategoryEntryLine(--id--,--data--,--category-->
```

```
>--)-----------------------------------------------------------><
```

For more information, see SetEntryLine.

9.4.4. GetCategoryListLine

```
>>-aCategoryDialog~GetCategoryListLine(--id--,--category--)---->
```

For more information, see GetListLine.

9.4.5. SetCategoryListLine

```
>>-aCategoryDialog~SetCategoryListLine(--id--,--data--,--category--)->
```

For more information, see SetListLine.

9.4.6. GetCategoryListWidth

```
>>-aCategoryDialog~GetCategoryListWidth(--id--,--category--)--->
```

For more information, see GetListWidth.

9.4.7. SetCategoryListWidth

```
>>-aCategoryDialog~SetCategoryListWidth(--id--,--scrollwidth--,--category--)->
```

For more information, see SetListWidth.

9.4.8. GetCategoryMultiList

```
>>-aCategoryDialog~GetCategoryMultiList(--id--,--category--)--->
```

For more information, see GetMultiList.

9.4.9. SetCategoryMultiList

```
>>-aCategoryDialog~SetCategoryMultiList(--id--,--data--,--category-->

>--)---------------------------------------------------------------><
```

For more information, see SetMultiList.

9.4.10. GetCategoryComboLine

```
>>-aCategoryDialog~GetCategoryComboLine(--id--,--category--)--->><
```

For more information, see GetComboLine.

9.4.11. SetCategoryComboLine

```
>>-aCategoryDialog~SetCategoryComboLine(--id--,--data--,--category-->

>--)---------------------------------------------------------------><
```

For more information, see SetComboLine.

9.4.12. GetCategoryRadioButton

```
>>-aCategoryDialog~GetCategoryRadioButton(--id--,--category--)-><
```

For more information, see GetRadioButton.

9.4.13. SetCategoryRadioButton

```
>>-aCategoryDialog~SetCategoryRadioButton(--id--,--data--,------>

>--category--)-------------------------------------------------><
```

For more information, see SetRadioButton.

9.4.14. GetCategoryCheckBox

```
>>-aCategoryDialog~GetCategoryCheckBox(--id--,--category--)---->><
```

For more information, see GetCheckBox.

9.4.15. SetCategoryCheckBox

```
>>-aCategoryDialog~SetCategoryCheckBox(--id--,--data--,--category-->

>--)-----------------------------------------------------------><
```

For more information, see SetCheckBox.

9.4.16. GetCategoryValue

```
>>-aCategoryDialog~GetCategoryValue(--id--,--category--)-------><
```

For more information, see GetValue.

9.4.17. SetCategoryValue

```
>>-aCategoryDialog~SetCategoryValue(--id--,--data--,--category--)-><
```

For more information, see SetValue.

9.4.18. GetCategoryAttrib

```
>>-aCategoryDialog~GetCategoryAttrib(--aname--,--category--)---><
```

For more information, see GetAttrib.

9.4.19. SetCategoryAttrib

```
>>-aCategoryDialog~SetCategoryAttrib(--attributename--,--category--)-><
```

For more information, see SetAttrib.

9.5. Combo Box Methods

The following sections describe the individual combo box methods.

9.5.1. AddCategoryComboEntry

```
>>-aCategoryDialog~AddCategoryComboEntry(--id--,--data--,--category-->

>--)----------------------------------------------------------><
```

For more information, see AddComboEntry.

Arguments:

The arguments are:

id

The ID of the combo box

data

The text string that is added to the combo box

category

The category page number where the combo box is located

Example:

The following example adds a text string to the list of the combo box 101 in the third category page.

```
MyCategoryDialog~AddCategoryComboEntry(101, "I'm one of the choices", 3)
```

9.5.2. InsertCategoryComboEntry

```
>>-aCategoryDialog~InsertCategoryComboEntry(--id--,--item--,--data--,-->

>--category--)------------------------------------------------><
```

For more information, see InsertComboEntry.

9.5.3. DeleteCategoryComboEntry

```
>>-aCategoryDialog~DeleteCategoryComboEntry(--id--,--index--,--->

>--category--)------------------------------------------------><
```

For more information, see method DeleteComboEntry.

9.5.4. FindCategoryComboEntry

```
>>-aCategoryDialog~FindCategoryComboEntry(--id--,--data--,------>

>--category--)-----------------------------------------------><
```

For more information, see FindComboEntry.

9.5.5. GetCategoryComboEntry

```
>>-aCategoryDialog~GetCategoryComboEntry(--id--,--index--)----->><
```

For more information, see GetComboEntry.

9.5.6. GetCategoryComboItems

```
>>-aCategoryDialog~GetCategoryComboItems(--id--,--category--)-->><
```

For more information, see GetComboItems.

9.5.7. GetCurrentCategoryComboIndex

```
>>-aCategoryDialog~GetCurrentCategoryComboIndex(--id--,--category-->

>--)---------------------------------------------------------><
```

For more information, see GetCurrentComboIndex.

9.5.8. SetCurrentCategoryComboIndex

```
>>-aCategoryDialog~SetCurrentCategoryComboIndex(--id--,--+-------+-->
                                                         +-index-+

>--,--category--)-------------------------------------------><
```

For more information, see SetCurrentComboIndex.

9.5.9. ChangeCategoryComboEntry

```
>>-aCategoryDialog~ChangeCategoryComboEntry(--id--,--item--,--data-->

>--,--category--)---------------------------------------------><
```

For more information, see ChangeComboEntry.

9.5.10. CategoryComboAddDirectory

```
>>-aCategoryDialog~CategoryComboAddDirectory(--id--,--drvpath--,-->

>--fileAttributes--,--category--)-----------------------------><
```

For more information, see ComboAddDirectory.

9.5.11. CategoryComboDrop

```
>>-aCategoryDialog~CategoryComboDrop(--id--,--category--)------><
```

For more information, see ComboDrop.

9.6. List Box Methods

The following sections describe the individual list box methods.

9.6.1. AddCategoryListEntry

```
>>-aCategoryDialog~AddCategoryListEntry(--id--,--data--,--category-->

>--)---------------------------------------------------------><
```

For more information, see AddListEntry.

9.6.2. InsertCategoryListEntry

```
>>-aCategoryDialog~InsertCategoryListEntry(--id--,--item--,--data-->

>--,--category--)---------------------------------------------><
```

For more information, see InsertListEntry.

9.6.3. DeleteCategoryListEntry

```
>>-aCategoryDialog~DeleteCategoryListEntry(--id--,--index--,---->

>--category--)-----------------------------------------------><
```

For more information, see DeleteListEntry.

9.6.4. FindCategoryListEntry

```
>>-aCategoryDialog~FindCategoryListEntry(--id--,--data--,--category-->

>--)---------------------------------------------------------><
```

For more information, see FindListEntry.

9.6.5. GetCategoryListEntry

```
>>-aCategoryDialog~GetCategoryListEntry(--id--,--ndx--,--category-->

>--)---------------------------------------------------------><
```

For more information, see GetListEntry.

9.6.6. GetCategoryListItems

```
>>-aCategoryDialog~GetCategoryListItems(--id--,--category--)--->
```

For more information, see GetListItems.

9.6.7. GetCurrentCategoryListIndex

```
>>-aCategoryDialog~GetCurrentCategoryListIndex(--id--,--category--)->
```

For more information, see GetCurrentListIndex.

9.6.8. SetCurrentCategoryListIndex

```
>>-aCategoryDialog~SetCurrentCategoryListIndex(--id--,--+-------+-->
                                                       +-index-+

>--,--category--)---------------------------------------------><
```

For more information, see SetCurrentListIndex.

9.6.9. ChangeCategoryListEntry

```
>>-aCategoryDialog~ChangeCategoryListEntry(--id--,--item--,--data-->

>--,--category--)---------------------------------------------><
```

For more information, see ChangeListEntry.

9.6.10. SetCategoryListTabulators

```
                                                  +-,---+
                                                  V     |
>>-aCategoryDialog~SetCategoryListTabulators(--id--,----tab-+--->

>--,--category--)---------------------------------------------><
```

For more information, see SetListTabulators.

9.6.11. CategoryListAddDirectory

```
>>-aCategoryDialog~CategoryListAddDirectory(--id--,--drvpath--,-->

>--fileAttributes--,--category--)-----------------------------><
```

For more information, see ListAddDirectory.

9.6.12. CategoryListDrop

```
>>-aCategoryDialog~CategoryListDrop(--id--,--category--)-------><
```

For more information, see ListDrop.

9.7. Appearance Modification Methods

The following sections describe the methods affecting the appearance of the item.

9.7.1. EnableCategoryItem

```
>>-aCategoryDialog~EnableCategoryItem(--id--,--category--)----->><
```

For more information, see EnableItem.

9.7.2. DisableCategoryItem

```
>>-aCategoryDialog~DisableCategoryItem(--id--,--category--)---->><
```

For more information, see DisableItem.

9.7.3. ShowCategoryItem

```
>>-aCategoryDialog~ShowCategoryItem(--id--,--category--)------->><
```

For more information, see ShowItem.

9.7.4. HideCategoryItem

```
>>-aCategoryDialog~HideCategoryItem(--id--,--category--)------->><
```

For more information, see HideItem.

9.7.5. SetCategoryItemFont

```
>>-aCategoryDialog~SetCategoryItemFont(--id--,--fonthandle--,--redraw--,-->

>--category--)------------------------------------------------><
```

For more information, see SetItemFont.

9.7.6. FocusCategoryItem

```
>>-aCategoryDialog~FocusCategoryItem(--id--,--category--)------><
```

For more information, see FocusItem.

9.7.7. ResizeCategoryItem

```
>>-aCategoryDialog~ResizeCategoryItem(--id--,--width--,--height--,-->

>--+----------------------+--,--category--)--------------------><
   +-"--+-HIDEWINDOW-+--"-+
        +-SHOWWINDOW-+
        +-NOREDRAW---+
```

For more information, see ResizeItem.

9.7.8. MoveCategoryItem

```
>>-aCategoryDialog~MoveCategoryItem(--id--,--xPos--,--yPos--,--->

>--+----------------------+--,--category--)--------------------><
   +-"--+-HIDEWINDOW-+--"-+
        +-SHOWWINDOW-+
        +-NOREDRAW---+
```

For more information, see MoveItem.

9.7.9. SendMessageToCategoryItem

```
>>-aCategoryDialog~SendMessageToCategoryItem(--id--,--msg--,--wp--,--lp-->

>--,--category--)---------------------------------------------><
```

For more information, see SendMessageToItem.

Chapter 10. Standard Dialogs, External Functions, and Public Routines

The ooDialog framework provides a number of easy to use standard dialogs, external functions, and public routines that allow programmers to include simple graphical elements in their ooRexx applications. A few of the functions, like MSSleep expose functionality that can be useful in any type of ooRexx application. The external functions and public routines can be called with a single line of code and the standard dialogs can usually be executed with only two lines of code. This does not require the programmer to do much set up and makes them simple to work with.

The standard dialogs, external functions, and public routines are listed the following table:

Table 10-1. Standard Dialogs, Functions, and Routines

Standard dialog...	**...on page**
CheckList	CheckList class
InputBox	InputBox class
IntegerBox	IntegerBox class
ListChoice	ListChoice class
MultiInputBox	MultiInputBox class
MultiListChoice	MultiListChoice class
PasswordBox	PasswordBox class
SingleSelection	SingleSelection class
TimedMessage	TimedMessage class
External function...	**...on page**
ErrorMessage	ErrorMessage function
GetFileNameWindow	GetFileNameWindow function
GetScreenSize	GetScreenSize function
GetSysMetrics	GetSysMetrics function
InfoMessage	InfoMessage function
PlaySoundFile	PlaySoundFile function
PlaySoundFileInLoop	PlaySoundFileInLoop function
SleepMS	SleepMS function
StopSoundFile	StopSoundFile function
WinTimer	WinTimer function
YesNoMessage	YesNoMessage function
Public routine...	**...on page**
AskDialog	AskDialog routine
CheckList	CheckList routine
ErrorDialog	ErrorDialog routine
FileNameDialog	FileNameDialog routine

Public routine...	...on page
FindWindow	FindWindow routine
InfoDialog	InfoDialog routine
InputBox	InputBox routine
IntegerBox	IntegerBox routine
ListChoice	ListChoice routine
MSSleep	MSSleep routine
MultiInputBox	MultiInputBox routine
MultiListChoice	MultiListChoice routine
PasswordBox	PasswordBox routine
Play	Play routine
ScreenSize	ScreenSize routine
SingleSelection	SingleSelection routine
SystemMetrics	SystemMetrics routine
TimedMessage	TimedMessage routine

10.1. DlgUtil Class

The DlgUitl class has a single class method, *version*. This method is used to determine the current executing version of ooDialog.

Requires:

The DlgUtil class requires the class definition file `oodPlain.cls`:

```
::requires "oodPlain.cls"
```

10.1.1. version (Class Method)

```
>>-.DlgUtil~version(--+----------+--)---------------------><
                      +-format---+
```

Returns the ooDialog version string.

Arguments:

The single argument is:

format [optional]

A keyword indicating the format of the returned version string. Only the first letter is needed and case is not significant. Any unrecognized keyword is ignored and the default format is used. The format for each keyword is as follows:

Full

A string specifying the version of ooDialog. This is the default. The string has the format: `ooDialog Version x.x.x.SS (an ooRexx Windows Extension)` where x.x.x is the ooDialog version and SS is the Subversion revision number of the build. The actual version string might look like: `ooDialog Version 4.1.0.8038 (an ooRexx Windows Extension)`

Short

Only the x.x.x.SS part of the full version string is returned.

Return value:

A string in one of the formats specified above.

Example:

```
v = .DlgUtil~version
say v
say

if .DlgUtil~version("SHORT") < 4.1.0.0000 then do
  say 'Too bad, you are using ooDialog 4.0.0'
end
else do
  say 'Great you have upgraded past ooDialog 4.0.0'
end

::requires 'oodPlain.cls'

/* On the screen you might see:

ooDialog Version 4.0.0.3768 (an ooRexx Windows Extension)

Too bad, you are still using ooDialog 4.0.0

*/
```

10.2. Standard Dialog Classes

The standard dialog classes are:

- TimedMessage
- InputBox
- PasswordBox

- IntegerBox
- MultiInputBox
- ListChoice
- MultiListChoice
- CheckList
- SingleSelection

Requires:

Use the `oodPlain.cls` file for the standard dialog classes.

```
::requires "oodPlain.cls"
```

Preparation:

Standard dialogs are prepared by using the `new` method of the class, which in turn invokes the `Init` method. The parameters for the `new` method are described for the `Init` method of each class.

Execution:

The dialog is then run by using the `Execute` method. For most of the standard dialogs `Execute` returns the user's input if the OK button is clicked and the null string if the Cancel button is clicked to terminate the dialog. If there is more than one return value, Execute returns the value 1 and stores the results in an attribute. Exceptions to the general rule are noted in the documentation for the individual dialog.

Functions:

Each standard dialog is also available as a callable function. These functions are described in the Public Routines section of this chapter.

10.2.1. TimedMessage Class

The TimeMessage class shows a message window for a defined duration.

Requires:

`oodPlain.cls` is required to use this class

Subclass:

This class is a subclass of the `PlainUserDialog` class

Execute:

There is no return from execute for this dialog.

The methods listed below are defined by this class.

10.2.1.1. Init

```
>>-aTimedMessage~Init(--msg--,--title--,--sleeping--+--------------+--)---->
                                                    +-,-earlyReply-+
```

The Init method prepares the dialog.

Arguments:

The arguments are:

message

A string that is displayed inside the window as a message. The length of the message determines the horizontal size of all standard dialogs.

title

A string that is displayed as the window title in the title bar of the dialog

sleeping

A number that determines how long (in milliseconds) the window is shown. If this number is 0 or greater, the message window is shown for that time and automatically dismisses itself. The programmer need take no further action.

If the number is less than 0 then the message window is shown indefinitely. The `Execute` method returns immediately. In this case, the programmer needs to dismiss the window manually. This is done by invoking the `stopIt` method. See **Example 2** below.

earlyReply

The optional early reply argument defaults to false. If used and if true, the `execute` method of the dialog returns immediately. This allows the code at the point where `execute` was invoked to continue. The message windows continues to display for its specified duration and then dismisses itself automatically.

Example 1:

The following example shows a window with the title of *Infomation* for a duration of 3 seconds:

```
dlg = .TimedMessage~New("Application will be started, please wait", -
                        "Information", 3000)
dlg~Execute
drop dlg
```

Example 2:

The following example shows an introductory window that displays while an application is doing its time consuming start up routine. Since this start up process can vary in time depending on the individual system, the window is set to display indefinitely. When the start up process is finished, the programmer dismisses the window and destroys the dialog manually.

```
dlg = .TimedMessage~New("The WidgetCreator Application - loading ...", -
                        "The Widget Factory", -1)
dlg~Execute
```

```
ret = doStartup()
dlg~stopIt
drop dlg
if ret <> 0 then do
  ...
end
```

Example 3:

The following example is a variation on **Example 2**. In this case the Widget Factory executives decided that they want their WidgeCreator splash screen to always display for 2 seconds to the customer and then close. The early reply argument is used so that the start up routine executes immediately. After 2 seconds the splash screen dismisses and the dialog is destroyed automatically.

> **Note:** It is not necessary to explicitly drop the dlg variable. That is shown in the other examples to emphasis the point that the dialog has been destroyed.

```
dlg = .TimedMessage~New("The WidgetCreator Application - loading ...", -
                        "The Widget Factory", 2000)
dlg~Execute
if doStartup() <> 0 then do
  ...
end
```

10.2.1.2. DefineDialog

```
>>-aTimedMessage~DefineDialog---------------------------------><
```

The DefineDialog method is called by the Create method of the parent class, PlainUserDialog, which in turn is called at the very beginning of Execute. You do not have to call it. However, you may want to override it in your subclass to add more dialog controls to the window. If you override it, you have to forward the message to the parent class by using the keyword super.

Example:

The following example shows how to subclass the TimedMessage class and how to add a background bitmap to the dialog window:

```
::class MyTimedMessage subclass TimedMessage inherit DialogExtensions

::method DefineDialog
   self~BackgroundBitmap("mybackg.bmp", "USEPAL")
   self~DefineDialog:super()
```

10.2.1.3. Execute

```
>>-aTimedMessage~Execute--------------------------------------><
```

The Execute method creates and shows the message window. If the specified duration (see Init method) is not negative, this method destroys the dialog automatically when the duration is up. If the duration is less than 0, then the programmer must destroy the dialog manually by invoking the `stopIt` method. See this example for a code snippet showing the procedure.

10.2.2. InputBox Class

The InputBox class provides a simple dialog with a title, a message, one entry line, an OK, and a Cancel push button.

Requires:

`oodPlain.cls` is required to use this class

Subclass:

This class is a subclass of the `PlainUserDialog` class

Execute:

Returns the user's input

The methods listed below are defined by this class.

10.2.2.1. Init

```
>>-aInputBox~Init(--message--,--title--,--prevalue--,--len--)---->><
```

The Init method prepares the input dialog.

Arguments:

The arguments are:

message

A text string that is displayed in the dialog

title

A string that is displayed as the dialog's title in the title bar

prevalue

A string to initialize the entry line. If you do not want to put any text in the entry line, just pass an empty string.

len

The width of the entry line in dialog units

Example:

The following example shows an InputBox dialog with the title of *Input* and an entry line:

```
dlg = .InputBox~New("Please enter your email address", -
                    "Input", "user@host.domain", 150)
value = dlg~Execute
say "You entered:" value
drop dlg
```

10.2.2.2. DefineDialog

```
>>-aInputBox~DefineDialog-------------------------------------><
```

The DefineDialog method is called by the Create method of the parent class, PlainUserDialog, which in turn is called at the very beginning of Execute. You do not have to call it. However, you may want to override it in your subclass to add more dialog controls to the window. If you override it, you have to forward the message to the parent class by using the keyword super.

10.2.2.3. AddLine

```
>>-aInputBox~AddLine(--x--,--y--,--l--)------------------------><
```

The AddLine method is used internally to add one entry line to the dialog.

10.2.2.4. Execute

```
>>-aInputBox~Execute------------------------------------------><
```

The Execute method creates and shows the dialog. After termination, the value of the entry line is returned if the user clicks the OK button; a null string is returned if the user clicks on Cancel.

10.2.3. PasswordBox Class

The PasswordBox class is an InputBox dialog with an entry line that echoes the keys with asterisks (*) instead of characters.

Requires:

`oodPlain.cls` is required to use this class

Subclass:

This class is a subclass of the InputBox Class

Execute:

Returns the user's password

The methods are the same as for the InputBox Class, with the exception of AddLine.

10.2.3.1. AddLine

```
>>-aPasswordBox~AddLine(--x--,--y--,--l--)--------------------><
```

The AddLine overrides the same method of the parent class, InputBox, by using a password entry line instead of a simple entry line.

10.2.4. IntegerBox Class

The IntegerBox class is an InputBox dialog whose entry line allows only numerical data.

Requires:

`oodPlain.cls` is required to use this class

Subclass:

This class is a subclass of the InputBox Class class

Execute:

Returns the user's numeric input

The methods are the same as for the InputBox Class class, with the exception of Validate.

10.2.4.1. Validate

```
>>-aIntegerBox~validate----------------------------------------><
```

The only method this subclass overrides is Validate, which is one of the automatically called methods of PlainUserDialog. It is invoked by the OK method, which in turn is called in response to a push button event. This method checks whether or not the entry line contains a valid numerical value. If the value is invalid, a message window is displayed.

10.2.5. MultiInputBox Class

The MultiInputBox class is a dialog that provides a title, a message, and one or more entry lines. After execution of this dialog you can access the values of the entry lines.

Requires:

`oodPlain.cls` is required to use this class

Subclass:

This class is a subclass of the "PlainUserDialog" class

Execute:

Returns 1 (if OK was clicked). The values entered by the user are stored in attributes matching the labels of the entry lines.

The methods are the same as for the InputBox Class class, with the exception of Init.

10.2.5.1. Init

```
>>-aMultiInputBox~Init(--message--,--title--,--labels.--,--datas.-->

>--+--------+--)-----------------------------------------------><
   +-,--len-+
```

The Init method is called automatically whenever a new instance of this class is created. It prepares the dialog.

Arguments:

The arguments are:

message

A text string that is displayed on top of the entry lines. Use it to give the user advice on what to do.

title

A text string that is displayed in the title bar.

labels.

A stem variable containing strings that are used as labels on the left side of the entry lines. Labels.1 becomes the label for the first entry line, labels.2 for the second, and so forth.

datas.

A stem variable (do not forget the trailing period) containing strings that are used to initialize the entry lines. The entries must start with 101 and continue in increments of 1.

len

The length of the entry lines. All entry lines get the same length.

Example:

The following example creates a four-line input box. The data entered is stored in the object attributes that are displayed after dialog execution.

```
lab.1 = "First name"  ;     lab.2 = "Last name "
lab.3 = "Street and City" ; lab.4 = "Profession:"

addr.101 = "John" ; addr.102 = "Smith" ; addr.103 = ""
addr.104 = "Software Engineer"

dlg = .MultiInputBox~new("Please enter your address", ,
"Your Address", lab., addr.)
if dlg~execute = 1 then do
  say "The address is:"
  say dlg~firstname dlg~lastname
  say dlg~StreetandCity
  say dlg~Profession
end
```

10.2.6. ListChoice Class

The ListChoice class provides a dialog with a list box, an OK, and a Cancel button. The selected item is returned if the OK push button is used to terminate the dialog.

Requires:

`oodPlain.cls` is required to use this class

Subclass:

This class is a subclass of the "PlainUserDialog" class

Execute:

Returns the user's choice or a null string

The method listed below is defined by this class.

10.2.6.1. Init

```
>>-aListChoice~Init(--message--,--title--,--input.-------------->

>--+-----------------------------------------------------+------><
   +-,--+------+--+-----------------------------+--)-+
        +-lenx-+  +-,--+------+--+--------------+-+
                       +-leny-+  +-,--preselect-+
```

The Init method is used to initialize a newly created instance of this class.

Arguments:

The arguments are:

message

A text string that is displayed on top of the list box. Use it to give the user advice on what to do.

title

A text string for the dialog's title

input.

A stem variable (do not forget the trailing period) containing string values that are inserted into the list box

lenx, leny

The size of the list box in dialog units

preselect

Entry that is selected when list pops up

Example:

The following example creates a list choice dialog box where the user can select exactly one dessert:

```
lst.1 = "Cookies"; lst.2 = "Pie"; lst.3 = "Ice cream"; lst.4 = "Fruit"

dlg = .ListChoice~new("Select the dessert please","YourChoice",lst., , ,"Pie")
say "Your ListChoice data:" dlg~execute
```

10.2.7. MultiListChoice Class

The MultiListChoice class is an extension of the ListChoice class. It makes it possible for the user to select more than one line at a time. The Execute method returns the selected items' indexes separated by blank spaces. The first item has index 1.

Requires:

`oodPlain.cls` is required to use this class

Subclass:

This class is a subclass of the ListChoice Class

Execute:

Returns the index numbers of the entries selected

Preselect:

Indexes of entries, separated by a blank, that are to be preselected. The first entry has index 1 and the rest are increments of one.

The methods are the same as for the ListChoice Class class, except that Execute returns the index numbers of the selected entries.

Example:

The following example creates a multiple list choice box where the user can select multiple entries:

```
lst.1 = "Monday" ;   lst.2 = "Tuesday" ; lst.3 = "Wednesday"
lst.4 = "Thursday" ; lst.5 = "Friday" ;  lst.6 = "Saturday"
lst.7 = "Sunday"

dlg = .MultiListChoice~new("Select the days you are working this week", ,
                           "YourMultipleChoice",lst., , ,"2 5")
s = dlg~execute
if s <> "" then do while s \= ""
                  parse var s res s
                  say lst.res
               end
```

10.2.8. CheckList Class

The CheckList class is a dialog with a group of one or more check boxes.

Requires:

`oodPlain.cls` is required to use this class.

Subclass:

This class is a subclass of the "PlainUserDialog" class.

Execute:

Returns 1 (if OK was clicked). The check boxes selected by the user are marked in a stem variable with the value 1.

The method listed below is defined by this class.

10.2.8.1. Init

```
>>-aCheckList~Init(--message--,--title--,--labels.--,--datas.-->

>>--+-------------------------+--)----------------------------><
    +-,--+-----+--+--------+-+
         +-len-+  +-,--max-+
```

Arguments:

The arguments are:

message

A text string that is displayed on top of the check box group. Use it to give the user advice on what to do.

title

A text string for the dialog's title

labels.

A stem variable (do not forget the trailing period) containing all the labels for the check boxes

datas.

This argument is a stem variable (do not forget the trailing period) that you can use to preselect the check boxes. The first check box relates to stem item 101, the second to 102, and so forth. A value of 1 indicates selected, and a value of 0 indicates deselected.

For example, Datas.103=1 indicates that there is a check mark on the third box.

len

Determines the length of the check boxes and labels. If omitted, the size is calculated to fit the largest label.

max

The maximum number of check boxes in one column. If there are more check boxes than max - that is, labels. has more items than the value of max - this method continues with a new column.

Example:

The following example creates and shows a dialog with seven check boxes:

```
lst.1 = "Monday";   lst.2 = "Tuesday"; lst.3 = "Wednesday"
lst.4 = "Thursday"; lst.5 = "Friday";  lst.6 = "Saturday"
lst.7 = "Sunday"

do i = 101 to 107
   chk.i = 0
end

dlg = .CheckList~new("Please select a day!","Day of week",lst., chk.)
if dlg~execute = 1 then do
   say "You selected the following day(s): "
   do i = 101 to 107
      a = i-100
      if chk.i = 1 then say lst.a
   end
end
```

10.2.9. SingleSelection Class

The SingleSelection class shows a dialog that has a group of radio buttons. The user can select only one item of the group.

Requires:

`oodPlain.cls` is required to use this class

Subclass:

This class is a subclass of the "PlainUserDialog" class

Execute:

Returns the number of the radio button selected

The method listed below is defined by this class.

10.2.9.1. Init

```
>>-aSingleSelection~Init(--message--,--title--,--labels.--,--data-->

>--+------------------------+--)-----------------------------><
   +-,--+-----+--+--------+-+
        +-len-+  +-,--max-+
```

Arguments:

The arguments are:

message

A text string that is displayed on top of the radio button group. Use it to give the user advice on what to do.

title

A text string for the title bar

labels.

This argument is a stem variable containing all labels for the radio buttons

data

You can use this argument to preselect one radio button. A value of 1 selects the first radio button, a value of 2 selects the second, and so on.

len

Determines the length of the check boxes and labels. If omitted, the size is calculated to fit the largest label.

max

The maximum number of radio buttons in one column. If there are more radio buttons than max - that is, labels. has more items than the value of max - this method continues with a new column.

Example:

The following example creates and executes a dialog that contains a two-column radio button group. The fifth radio button (the button with the label May) is preselected.

```
mon.1 = "January" ; mon.2 = "February" ; mon.3 = "March"
mon.4 = "April"   ; mon.5 = "May"      ; mon.6 = "June"
mon.7 = "July"    ; mon.8 = "August"   ; mon.9 = "September"
mon.10= "October" ; mon.11= "November" ; mon.12= "December"

dlg = .SingleSelection~new("Please select a month!", ,
                           "Single Selection", mon., 5, , 6)
s = dlg~execute
say  "You Selected the month: " mon.s
```

10.3. External Functions

The ooDialog framework provides external functions that can be used as callable functions in your Open Object Rexx programs. These functions are registered automatically when the first dialog is initialized. If no dialog has been created, the programmer must register the functions prior to calling them. An individual function is registered with:

```
call RxFuncAdd functionname, "oodialog", functionname
```

To register all the ooDialog functions in this group use:

```
call RxFuncAdd InstMMFuncs, "oodialog", InstMMFuncs
call InstMMFuncs
```

A more convenient way to call many of these functions is provided by the Public Routines in the ooDialog framework. The public routines do not require that the programmer register any of the external functions. That task is handled transparently by the routine.

Note: The ooDialog framework includes a number of external functions that are not listed here. In general, using the external functions not listed requires a good working knowledge of C and an understanding of the underlying Windows API. Therefore, those functions are not listed. The functions listed in this section are the functions that are convenient and easy to use in an ooRexx program for the general Rexx programmer. They require no understanding of C or the Windows API.

10.3.1. InfoMessage

Displays an information message window:

```
call InfoMessage "some message text"
ret = InfoMessage("another text")
```

See also the public routine InfoDialog.

10.3.2. ErrorMessage

Displays an error message window:

```
call ErrorMessage "some error message text"
ret = ErrorMessage("another error message")
```

See also the public routine ErrorDialog.

10.3.3. YesNoMessage

Displays a message and ask the user for a YES or NO answer:

```
ret = YesNoMessage("Delete these files?", "No")
if ret=1 then /* this is yes */
```

The second argument is optional and can have a value of either Yes or No. An argument of Yes gives the Yes button the default focus, No gives the No button the default focus. If the argument is omitted the Yes button is given the default focus. Only the first letter of the argument is needed and case is not significant.

See also the public routine AskDialog.

10.3.4. GetScreenSize

Queries the monitor size in dialog units and pixels:

```
val = GetScreenSize()
parse var val dunitx dunity pixelx pixely
```

See also the public routine ScreenSize.

10.3.5. GetSysMetrics

Queries a system metric value:

```
SM_CMONITORS = 80
metric = GetSysMetric(SM_CMONITORS)
say 'There are' metric 'monitors connected to this system.'
```

The public routine, SystemMetrics, provides an easy way to use this external routine without first creating an ooDialog dialog or manually loading the C functions. Check the documentation for the public routine for more detailed information on the system metrics in Windows.

10.3.6. PlaySoundFile

Plays a sound file (.WAV):

```
call PlaySoundFile "d:\wav\sound.wav"
ret = PlaySoundFile("d:\wav\sound.wav","YES")
```

The optional second parameter YES plays the file asynchronously, that is the program continues execution. See also the public routine Play.

10.3.7. PlaySoundFileInLoop

Plays a sound file (.WAV) continuously and asynchronously:

```
call PlaySoundFileInLoop "d:\wav\sound.wav"
```

10.3.8. StopSoundFile

Stops playing an asynchronous sound file (.WAV):

```
call StopSoundFile
```

10.3.9. GetFileNameWindow

Displays an Open File window:

```
file = GetFileNameWindow(filename,handle,filter,loadorsave,title, ,
                         defExtension,multiSelect,sepChar)
```

Parameters (optional):

filename

A preselected file name and / or initial directory information.

This argument places a preselected file name in the "File name" field of the file selection dialog. In addition this argument can also be used to specify the initial directory (folder) the dialog opens in.

When this argument contains no path information, then the File name field will contain the argument and the initial directory is determined by the operating system. Which directory the operating system picks varies slightly between Windows 98, Windows NT, Windows XP, etc. But, in general, it will be a directory previously used in one of the Windows common dialogs, or a directory in the users My Documents. (The exact behavior is documented by Microsoft.)

If the argument contains file name and path information, the initial directory will be determined by the path information and the File name field will contain the file information. When the filename argument contains only path information and no file name information, i.e. the argument ends with a back slash ("\"), then the initial directory will be that path and the File name field will be left blank.

In cases where the argument contains incorrect path information the initial directory is determined as if no path information had been supplied and the entire value of the argument is placed in the File name field.

> **Note:** Windows treats normal wild card characters as valid for file names. Therefore, values such as C:*.* or C:\Windows*.exe are acceptable for this argument.

handle

The parent window handle.

filter

A file mask specification.

loadorsave

If you specify 1, which is the default, you get the file name for a load operation. If you specify 0, you get the file name for a save operation.

title

The window title. The default is "Open a File" or "Save File As", depending on what you specify for *loadorsave*.

defExtension

The default extension that is added if no extension was specified. The default is TXT.

multiSelect

If you specify 1, you can select several files. In this case, *loadorsave* must also be 1. The result is then *path file1 file2 file3 ....*

If you specify 0 or omit this parameter, you get the selected file name or an empty string when the Open File window is canceled.

sepChar

Specifies which character should be used for separating the file names when *multiSelect* = 1.

This is needed for file names with blank characters. If this argument is omitted, the separation character is a blank. If the argument is specified, the returned path and file name uses this separation character. For example, if you specify "#" as the separation character, the return string might look as follows:

```
C:\WINNT#file with blank.ext#fileWithNoBlank.TXT
```

Example:

```
"Text files (*.txt)"'0'x"*.TXT"'0'x ,
"All files (*.*)"'0'x"*.*"
```

10.3.10. SleepMS

Sleeps for a given time interval, in milliseconds:

```
call SleepMS(3000) /* 3 seconds */
```

See also the public routine MSSleep.

10.3.11. WinTimer

Starts, stops, and waits for a windows timer:

```
tid = WinTimer("START",300)   /* 0,3 seconds */
call  WinTimer("WAIT,tid)     /* wait... */
ret = WinTimer("STOP",tid)    /* stop premature */
```

10.4. Public Routines

The routines listed in this section can be used in any ooRexx program. They are designed to be even easier to use than the standard dialogs or the external functions. They do not require that an ooDialog dialog be initialized prior to their use, or that any ooDialog dialog be used in the program at all. The programmer does not have to register any functions to use these routines.

Requires:

Use a requires directive for the `ooDialog.cls` file in an ooRexx program to use any of these routines.

```
::requires ooDialog.cls
```

10.4.1. Play Routine

This routine is used to play audio sounds.

```
>>-Play(--+----------+--+---------+--)------------------------><
          +-fileName-+  +-,--YES--+
                        +-,--LOOP-+
```

The Play routine can be used to play an audio file using the Windows multimedia capabilities. See also the PlaySoundFile external function.

The named file is searched for in the current directory and in the directories of the SOUNDPATH environment variable.

Arguments:

The arguments are:

fileName

The file name of an audio (.WAV) file. The file is searched for in the current directory and in the directories listed in the SOUNDPATH environment variable. If this argument is omitted, the currently playing sound file is stopped.

option

You can set the last argument to:

YES

This plays the audio file asynchronously.

LOOP

Plays the audio file asynchronously and repeats it in a loop. You can stop the loop by calling Play again omitting all arguments.

Example:

The following example plays a welcoming message:

```
rc = play("Welcome.wav")
```

10.4.2. InfoDialog Routine

Pops up a message box containing the specified text and an OK button.

```
>>-InfoDialog(--info_text--)---------------------------------><
```

Argument:

The only argument is:

info_text

Text to be displayed in the message box.

10.4.3. ErrorDialog Routine

Pops up a message box containing the specified text, an OK button, and an error symbol.

```
>>-ErrorDialog(--error_text--)-------------------------------><
```

Argument:

The only argument is:

error_text

Text to be displayed in the message box.

10.4.4. AskDialog Routine

Pops up a message box containing the specified text, a Yes button, and a No Button.

```
>>-AskDialog(--question--+-------------------+--)-------------><
                        +--, defaultbutton--+
```

Arguments:

The only argument is:

question

Text to be displayed in the message box.

defaultbutton

Specifies which button, the Yes or the No button, has the default focus when the message box pops up. This argument can be Yes or No and is optional. If the argument is omitted, the Yes button will be given the focus. Only the first letter of the option is needed and case is not significant.

Return Values:

0

The No button has been selected.

1

The Yes button has been selected.

10.4.5. FileNameDialog Routine

Causes a file selection dialog box to appear.

```
>>-FileNameDialog(--+---------+--+--------------------------------+-->
                    +-selfile-+  +-,--+--------+--+-------------+-+
                                       +-parent-+  +-,--filemask-+

>--+-------------------------------------------------------------->
   +-,--+------------+--+----------+--+-----------------+------------>
        +-loadorsave-+  +-,--title-+  +-,--defExtension-+


>--+-----------------------------------+--)------------------------><
>--+----------------+--+------------+--+
   +-,--multiSelect-+  +-,--sepChar-+
```

Arguments:

The arguments are:

selfile

Preselected file name and / or initial directory information.

This argument places a preselected file name in the "File name" field of the file selection dialog. In addition this argument can also be used to specify the initial directory (folder) the dialog opens in.

When this argument contains no path information, then the File name field will contain the argument and the initial directory is determined by the operating system. Which directory the operating system picks varies slightly between Windows 98, Windows NT, Windows XP, etc.

But, in general, it will be a directory previously used in one of the Windows common dialogs, or a directory in the users My Documents. (The exact behavior is documented by Microsoft.)

If the argument contains file name and path information, the initial directory will be determined by the path information and the File name field will contain the file information. When the selfile argument contains only path information and no file name information, i.e. the argument ends with a back slash ("\"), then the initial directory will be that path and the File name field will be left blank.

In cases where the argument contains incorrect path information the initial directory is determined as if no path information had been supplied and the entire value of the argument is placed in the File name field.

Note: Windows treats normal wild card characters as valid for file names. Therefore, values such as C:*.* or C:\Windows*.exe are acceptable for this argument.

parent

Handle to the parent window.

filemask

Pairs of null-terminated filter strings.

The first string in each pair is a display string that describes the filter (for example, "Text Files"), and the second string specifies the filter pattern (for example, "*.TXT"). To specify multiple filter patterns for a single display string, use a semicolon to separate the patterns (for example, "*.TXT;*.DOC;*.BAK"). A pattern string can be a combination of valid file name characters and the asterisk (*) wildcard character. Do not include spaces in the pattern string.

If omitted, `"Text Files (*.TXT)"'0'x"*.TXT"'0'x"All Files (*.*)"'0'x"*.*"` is used as the filter.

loadorsave

Specifies which dialog should be displayed, the File Open Dialog (the default) or the File Save Dialog. If the File Save Dialog is not explicitly requested the default File Open Dialog is always presented. Only the first letter of the argument is needed and case is not significant. For historic reasons this argument can also be 1 (File Open Dialog) or 0 (File Save Dialog.)

Load or 1

Display the File Open Dialog (default).

Save or 0

Display the File Save Dialog.

title

The window title. The default is "Open a File" or "Save File As", depending on what is specified for `loadorsave`.

defExtension

The default extension that is added if no extension was specified. The default is TXT.

multiSelect

Specifies if the File Open Dialog allows selection of multiple files. Normally, the open dialog allows the user to select one file only. This can be changed by specifying "MULTI" for this option. Only the first letter is needed and case is not significant.

MULTI

Multiple file selection is allowed. This option is ignored for File Save Dialogs. If the user selects multiple files, the result is then path file1 file2 file3 ...

sepChar

Specifies the separation character for the returned path and file names. This is needed for file names with blank characters. If this argument is omitted, the separation character is a blank. If the argument is specified, the returned path and file name uses this separation character. For example, if you specify "#" as the separation character, the return string might look as follows:

```
C:\WINNT#file with blank.ext#fileWithNoBlank.TXT
```

Return value:

Returns the selected file name or an empty string when canceled. Open File dialogs with multiple selection return path file1 file2 file3 ... when the user selects multiple files.

10.4.6. FindWindow Routine

Searches the Windows application list for a specific window and returns its handle.

```
>>-FindWindow(--Caption--)------------------------------------><
```

Argument:

The only argument is:

Caption

Caption of the window that is to be searched.

Return value:

Handle of the window or 0 if the window was not found.

10.4.7. ScreenSize Routine

Obtains the screen size in dialog units and in pixels

```
>>-ScreenSize()----------------------------------------------><
```

Argument:

This function takes no arguments.

Return value:

An array containing the screen size. The size in dialog units of the width is at index 1 and the height in dialog units is at index 2. The size in pixels of the width is at index 3 and the height in pixels is at index 4.

Example

Get the screen size and display it to the user.

```
size = screenSize()
say
say 'The monitor is' size[1] 'dialog units wide and' size[2] 'dialog units high.'
say 'The monitor is' size[3] 'pixels wide and' size[4] 'pixels high.'
```

10.4.8. SystemMetrics Routine

Queries and returns one of the system metric values The system metric values contain information that help the programmer write programs that work correctly independent of the current system. They have information on system configuration, such as the number of monitors connected, and metrics such as the size in pixels of a dialog border or the width in pixels of a scrollbar.

Good documentation on the system metrics is found in the MSDN Library under the GetSystemMetrics function. This documentation contains both the numeric value of the different indexes and information on what is returned for each index. The documentation is easy to found using a Google search of "GetSystemMetrics MSDN Library"

```
>>-SystemMetrics(--index--)----------------------------------><
```

Argument:

The only argument is:

index

The numeric index of the metric to query.

Return value:

The return value is dependent on the index queried. Consult the MSDN library for information on the returned values.

Note: The OS will return 0 if an invalid index is used. Hoever, the return value for some indexes is also 0. The programmer will need to determine from the context if 0 is an error return or not. In addition, ooDialog will return -1 for some invalid argument values. For instance, if the string "dog" was passed in for the `index` argument, -1 would be returned.

Example

Determine the number of mouse buttons and whether the mouse has a scroll wheel.

```
/* detectMouse.rex  Simple program to detect the user's mouse. */

SM_CMOUSEBUTTONS     = 43
SM_MOUSEWHEELPRESENT = 19

countMouseButtons = SystemMetrics(SM_CMOUSEBUTTONS)
haveMouseWheel = SystemMetrics(SM_MOUSEWHEELPRESENT)

if countMouseButtons == 0 then do
  say 'The system does not have a mouse attached'
end
else do
  say 'The mouse has' countMouseButtons 'buttons'
  if haveMouseWheel == 0 then say 'There is no mouse scroll wheel.'
  else say 'The mouse has a scroll wheel.'
end

return 0
::requires 'oodPlain.cls'
```

10.4.9. MSSleep Routine

Sleeps for the specified number of milliseconds.

```
>>-MSSleep(--duration--)-------------------------------------><
```

Argument:

The only argument is:

duration

The time in milliseconds to sleep.

Return value:

This function always returns 0.

10.4.10. TimedMessage Routine

The TimedMessage routine provides a shortcut to invoke a TimedMessage dialog as a function:

```
ret = TimedMessage("We are starting...","Please wait",3000)
```

The parameters are the same as described in the Init method of the `TimedMessage` class.

The function returns 0, always, when the duration is non-negative. When the duration is less than 0, the function returns a reference to the `TimedMessage` dialog object. This is needed in order for the programmer to manually dismiss the dialog. Below are the same 3 examples listed for the `TimedMessage` dialog, modified to work with the public routine:

Example 1:

The following example shows a window with the title of *Infomation* for a duration of 3 seconds:

```
ret = timedMessage("Application will be started, please wait", -
                   "Information", 3000)
```

Example 2:

The following example shows an introductory window that displays while an application is doing its time consuming start up routine. Since this start up process can vary in time depending on the individual system, the window is set to display indefinitely. When the start up process is finished, the programmer dismisses the window and destroys the dialog manually. Note how a reference to the `TimedMessage` dialog is returned. The programmer only needs to use this to dismiss / destroy the dialog

```
dlg = timedMessage("The WidgetCreator Application - loading ...", -
                   "The Widget Factory", -1)
ret = doStartup()
dlg~stopIt
if ret <> 0 then do
  ...
end
```

Example 3:

The following example is a variation on **Example 2**. In this case the Widget Factory executives decided that they want their WidgeCreator splash screen to always display for 2 seconds to the customer and then close. The early reply argument is used so that the start up routine executes immediately. After 2 seconds the splash screen dismisses and the dialog is destroyed automatically.

```
ret = timedMessage("The WidgetCreator Application - loading ...", -
                   "The Widget Factory", 2000)

if doStartup() <> 0 then do
```

```
    ...
end
```

10.4.11. InputBox Routine

A shortcut function to invoke an InputBox dialog as a function:

```
say "Your name:" InputBox("Please enter your name","Personal Data")
```

The parameters are described in the Init method of the`InputBox` class.

10.4.12. PasswordBox Routine

A shortcut to invoke a PasswordBox dialog as a function:

```
pwd = PasswordBox("Please enter your password","Security")
```

The parameters are described in the Init method of the `InputBox`class.

10.4.13. IntegerBox Routine

A shortcut to invoke an IntegerBox dialog as a function:

```
say "Your age:" IntegerBox("Please enter your age","Personal Data")
```

The parameters are described in the Init method of the `InputBox`class.

10.4.14. MultiInputBox Routine

Provides a shortcut function to invoke a MultiInputBox dialog:

```
res = MultiInputBox("Enter your address","Personal Data", ,
                    .array~of("&First name","Last &name","&City"), ,
                    .array~of("John","Smith","San Jose"), 100)
if res \= .Nil then do entry over res
                       say "Address-line[]= " entry
                    end
```

The parameters are described in the Init method of the `MultiInputBox`class. However, instead of using stems, arrays are used for input.

The user's input to the dialog is also returned in an array. Note this other difference from the MultiInputBox dialog: if the user cancels the dialog `.nil` is returned rather than the empty string.

10.4.15. ListChoice Routine

The ListChoice function provides a shortcut to invoke a ListChoice dialog:

```
day = ListChoice("Select a day","My favorite day", ,
                 .array~of("Monday","Tuesday","Wednesday","Thursday", ,
                           "Friday","Saturday","Sunday") , , ,"Thursday")
say "Your favorite day is" day
```

The parameters are described in the Init method of the `ListChoice`class. However, instead of an input stem an array is passed into the function.

If the user cancels the dialog `.nil` is returned rather than the empty string.

10.4.16. MultiListChoice Routine

Provides a shortcut to invoke a MultiListChoice dialog as a function:

```
days = MultiListChoice("Select days","My TV Days", ,
                       .array~of("Monday","Tuesday","Wednesday", ,
                              "Thursday","Friday","Saturday","Sunday"), , ,"2 5")
if days <> .Nil then do day over days
                       say "TV day =" day
                    end
```

The parameters are described in the Init method of the `MultiListChoice`class. However, instead of stems, arrays are passed into the function.

If the user cancels the dialog `.nil` is returned rather than the empty string.

10.4.17. CheckList Routine

A shortcut function to invoke the CheckList dialog:

```
>>--CheckList(--message--,--title--,--labels--,-+-----------+---->
                                               +-,--checks-+

>>--+-----------------------+--)------------------------------><
    +-,--+-----+--+--------+-+
         +-len-+  +-,--max-+
```

Arguments:

The arguments are similar to what is described in the Init method of the `CheckList`class. However, instead of stems, arrays are passed into and returned from the function.

message

A text string that is displayed on top of the check box group. It could be used, for example, to give the user a hint as to what to do.

title

A text string for the dialog's title

labels

An array containing the labels, in order, for each of the check boxes. The dialog will contain a check box for each label.

checks

This argument is an array that allows you to pre-check any of the check boxes. For each index in this array whose item is `.true`, (or 1,) the check box at the corresponding index in the `labels` array will be checked. For any index that is not `.true` , the corresponding check box will not be checked. This means that the programmer only needs to put a `.true` at the indexes where he wants the check boxes checked. All the other indexes can be left empty.

len

Determines the length (in dialog units) of the check boxes and labels. If omitted, the size is calculated to fit the largest label.

max

The maximum number of check boxes in one column. If there are more check boxes than max, that is, if labels has more items than the value of max, the check boxes will be put into columns. Each column will contain no more than the number of check boxes specified by max.

Return value:

If the user cancels the dialog, then `.nil` is returned. If the user clicks the okay button, then an array is returned. The array will be the same size as the input `labels` array. Each index of the returned array will contain `.true` if the corresponding check box was checked by the user or will contain `.false` if the check box was not checked.

Example

The following example show how to use the `CheckList` public routine:

```
weekdays = .array~of("Monday","Tuesday","Wednesday","Thursday", -
                     "Friday","Saturday","Sunday")

-- Monday and Tuesday will be checked, all the rest will not be checked.
checks = .array~of(.true, .true, .false)
checks[5] = 0
checks[7] = 'a'

-- The labels will be 60 dialog units wide and there will be at most 4 check
-- boxes in a column.  (Only 7 check boxes, so 2 columns, the first with 4
-- check boxes, the second with 3.)

days = CheckList("Check the days", "Working Days", weekdays, checks, 60, 4)

if days <> .Nil then do i = 1 to days~items
  if days[i] then say "Working day =" weekdays[i]
```

```
      end

    ::requires "oodPlain.cls"
```

10.4.18. SingleSelection Routine

The SingleSelection function provides a shortcut means to invoke a SingleSelection dialog

```
months = .array~of("Jan","Feb","Mar","Apr","May","Jun", ,
                   "Jul","Aug","Sep","Oct","Nov","Dec")
m = SingleSelection("Check it","Born in",months,12, ,6)
say "Born in month" m "=" months[m]
```

The parameters are described in the Init method of the `SingleSelection`class. However, instead of an input stem an array is passed into the function.

Chapter 11. DlgArea and DlgAreaU Classes

The DlgArea class provides assistance in laying out the dialog controls in a dynamically defined dialog. The DlgAreaU class provides assistance in resizing and / or positioning controls when a dialog is resized. Used together, the two classes provide a convenient way to create resizable dialogs.

11.1. DlgArea Class

The DlgArea class defines an area of a UserDialog and provides co-ordinates of and within it. It is a helper for the DefineDialog Method of an OODialog UserDialog Object and has no effect on any Windows Object.

To use objects of the DlgArea class include this line in your code:

```
::requires "OODIALOG.CLS"
```

The following figure shows the measuement attributes used to position a DlgArea on a UserDialog.

Figure 11-1. DlgArea Measurements

11.1.1. Init

```
>>- aDlgArea~Init(--X--,--Y--,--Width--,--Height--+-------------+--)-->&lt;
                                                  +--,--Margin--+
```

Arguments:

The arguments you pass to the new method when creating a DlgArea Object are:

X

The X Co-ordinate in dialog units relative to the UserDialog of the top left corner of the area you wish to define

Y

The Y Co-ordinate in dialog units relative to the UserDialog of the top left corner of the area you wish to define.

Width

The width in dialog units of the area of the UserDialog you wish to define

Height

The height in dialog units of the area of the UserDialog you wish to define

Margin

An inner margin within the DlgArea in Dialog Units to be left blank. The default is 5

Example:

```
u = .dlgArea~new(0, 0, self~SizeX,Self~SizeY)          /* u is whole of UserDialog */
b = .dlgArea~new(u~x("70%"), u~y , u~w("R"), u~h("R")) /* sub area for buttons */
```

11.1.2. B

Returns the y coordinate of the bottom margin in dialog units relative to UserDialog.

```
>>--ADlgArea~B-------------------------------------------------------><
```

11.1.3. Bottom

Returns the y coordinate of the bottom edge in dialog units relative to UserDialog.

```
>>--ADlgArea~Bottom--------------------------------------------------><
```

11.1.4. CX

Synonym for the W method.

```
>>--ADlgArea~cx(--+-------+--)----------------------------------------><
                  +--n%---+
                  +--"R"--+
```

11.1.5. CY

Synonym for the H method.

```
>>--ADlgArea~cy(--+-------+--)----------------------------------------><
                  +--n%---+
                  +--"R"--+
```

11.1.6. H

Returns a height in dialog units relative to the dialog area.

If no argument is passed returns the inter-margin height.

```
>>--ADlgArea~h(--+-------+--)----------------------------------------><
                 +--n%---+
                 +--"R"--+
```

Arguments:

The arguments you pass to the H method are:

`n%`

If `n` is a percentage returns `n%` of the inter-margin height.

"R"

If "R" is passed returns Remaining height between last ~y and bottom margin.

11.1.7. HR

Returns the remaining height between last ~Y and bottom margin in dialog units. This is equivalent to ~H("R").

```
>>--ADlgArea~hr-------------------------------------------------------><
```

11.1.8. L

Returns the x coordinate of the left margin in dialog units relative to UserDialog.

```
>>--ADlgArea~L-------------------------------------------------------><
```

11.1.9. Left

Returns the x coordinate of the left edge in dialog units relative to UserDialog.

```
>>--ADlgArea~Left----------------------------------------------------><
```

11.1.10. Margin

Size in dialog units of DlgArea margin.

```
>>--ADlgArea~Margin--------------------------------------------------><
```

11.1.11. R

Returns the x coordinate of the right margin in dialog units relative to UserDialog.

```
>>--ADlgArea~R-------------------------------------------------------><
```

11.1.12. Right

Returns the x coordinate of the right edge in dialog units relative to UserDialog.

```
>>--ADlgArea~Right---------------------------------------------------><
```

11.1.13. T

Returns the y coordinate of the top margin in dialog units relative to UserDialog.

```
>>--ADlgArea~T-------------------------------------------------------><
```

11.1.14. Top

Returns the y coordinate of the top edge in dialog units relative to UserDialog.

```
>>--ADlgArea~Top-------------------------------------------------------><
```

11.1.15. W

Returns a width in dialog units relative to the dialog area.

If no argument is passed returns the inter-margin width.

```
>>--ADlgArea~w(--+-------+--)-----------------------------------------><
                 +--n%---+
                 +--"R"--+
```

Arguments:

The arguments you pass to the Y method are:

n%

If `n` is a percentage returns `n%` of the inter-margin width.

"R"

If "R" is passed returns Remaining width between last ~x and right margin.

11.1.16. WR

Returns the remaining width between last ~X and right margin in dialog units. This is equivalent to ~W("R").

```
>>--ADlgArea~wr-------------------------------------------------------><
```

11.1.17. X

Returns an X Coordinate in dialog units relative to the dialog area.

If no `n` is passed returns the x coordinate of the left margin.

```
>>--ADlgArea~x(--+------+--)-----------------------------------------><
                 +--n---+
                 +--n%--+
```

Arguments:

The arguments you pass to the X method are:

n

If `n` is passed and is positive, then returns the x coordinate `n` dialog units to the right of the left margin. If `n` is passed and is negative, then returns the x coordinate `n` dialog units to the left of the right margin..

n%

If `n` is passed and is a percentage, then returns the x coordinate `n%` of the way between the left and right margins.

11.1.18. Y

Returns an Y Coordinate in dialog units relative to the dialog area.

If no `n` is passed returns the x coordinate of the top margin.

```
>>--ADlgArea~y(--+------+--)-----------------------------------------><
                 +--n---+
                 +--n%--+
```

Arguments:

The arguments you pass to the Y method are:

n

If `n` is passed and is positive, then returns the y coordinate `n` dialog units below the topmargin. If `n` is passed and is negative, then returns the y coordinate `n` dialog units above the bottom margin..

n%

If `n` is passed and is a percentage, then returns the y coordinate `n%` of the way between the top and bottom margins.

11.1.19. DlgArea Example

On a UserDialog we want the top left to be an entry area, with an area for buttons on the right and a status area below.

We decide to give 70% of the width and 90% of the height to the entry area. We leave the margin as the default. We want all the areas to resize themselves if we change the UserDialog size except the OK button should remain 15 units high.

Figure 11-2. DlgArea Plan

Here is what our DefineDialog method might look like:

```
/* ------------------------------------------------------------------------ */
::method DefineDialog
/* ------------------------------------------------------------------------ */
/* define DlgArea named u as whole of user dialog                           */
u=.dlgArea~new(  0       ,         0,self~SizeX,Self~SizeY)   /* whole dlg   */

/* define DlgArea named e within DlgArea u for entry line                   */
e=.dlgArea~new(u~x       ,u~y       ,u~w("70%"),u~h("90%"))

/* define DlgArea named s within DlgArea u for Status Area in remaining height*/
s=.dlgArea~new(u~x       ,u~y("90%"),u~w("70%"),u~hr      )

/* define DlgArea named b within DlgArea u for Button area                  */
b=.dlgArea~new(u~x("70%"),u~y       ,u~wr      ,u~hr      )

/* entry line coterminous with area e margins                               */
self~AddEntryLine(12,"text",e~x,e~y,e~w,e~h,"multiline")

/* Status Area Coterminous with area s margins                              */
self~AddText(s~x,s~y,s~w,s~h,"Status info appears here",11)

/* Seven buttons evenly spaced at 10% intervals, 9% high                    */
do i = 0 to 6
   self~addButton(12+i,b~x,b~y((i * 10)||"%"),b~w,b~h("9%"),"Button" i,"Button"||i)
end /* DO */

/* ok button 15 dialog units high at bottom of area b                       */
self~AddButton(1,b~x,b~y(-15),b~w,15,"OK","OK","Default")
```

Here is the resultant dialog.

Figure 11-3. Sample DlgArea

11.2. DlgAreaU Class

The DlgAreaU class is a subclass of the DlgArea class that assists the creation of dynamically resizable Dialogs.

To use objects of the DlgAreaU class include this line in your code:

```
::requires "OODIALOG.CLS"
```

11.2.1. Init

```
>>- aDlgAreaU~Init(Dialog--+-------------------------+--------+-)-->&lt;
                           +--,--Margin-+            |        |
                           +--,---------+--,-NoResize-+        |
                           +--,---------+--,----------+-NoMove-+
```

The DlgAreaU object creates a dialog area coterminous with the calling dialog. It inherits all the methods and attributes of the DlgArea Class (listed above).

Arguments:

The arguments you pass to the new method when creating a DlgArea Object are:

Dialog

An Object that is SubClassed from A UserDialog.

Margin

See DlgArea Margin.

NoResize

A set of Dialog Ids of widgets not to be resized during a resize.

NoMove

A set of Dialog Ids of widgets not to be moved during a resize.

11.2.2. CorrectionFactor

Adjusts the ratio of top & left margins to bottom & right (default=1.05) Creating resizable Dialogs:

```
>>--ADlgAreaU~CorrectionFactor----------------------------------------><
```

11.2.3. LastError

.nil or details of error parsing calling dialog's DefineDialog method.

```
>>--ADlgAreaU~LastError-----------------------------------------------><
```

11.2.4. NoMove

A set of Dialog Ids of widgets not to be moved during a resize.

```
>>--ADlgAreaU~NoResize------------------------------------------------><
```

11.2.5. NoResize

A set of Dialog Ids of widgets not to be resized during a resize.

```
>>--ADlgAreaU~NoResize------------------------------------------------><
```

11.3. Creating Resizeable Dialogs

You can use the DlgAreaU Object to facilitate in creating dynamically resizable Dialog objects, but to do so you must carefully satisfy all the requirements below.

The Dialog must be created with the `ThickFrame` option. It may be created with the UserDialog Create method or CreateCenter method.

Your Dialog init method must include the line:

```
self~connectResize("OnResize")
```

Your DefineDialog method must start with these lines:

```
expose u
u = .dlgAreaU~new(self)          /* whole dlg   */
```

Your DefineDialog method must not reference variables within the Add Method parameters, although you can use references to DlgArea attributes, so

```
Self~AddButton(23,1,5,6,"MyButton","Pressed")
self~addButton(14,b~x,b~y("10%"),b~w,b~h("9%"),"Button" 1,"Button"||1)
```

Both are OK, whereas

```
/* Seven buttons evenly spaced at 10% intervals, 9% high                    */
do i = 0 to 6
   self~addButton(12+i,b~x,b~y((i * 10)||"%"),b~w,b~h("9%"),"Button" i,"Button"||i)
end /* DO */
```

would now fail as the variable `i` is referenced within the parameters.

To debug your DlgAreaU call/insert the following after the instantiation

```
if u~lastError \= .nil then call errormessage u~lastError
```

You must also include the following in your code.

```
/* ------------------------------------------------------------------------ */
::method OnResize
/* ------------------------------------------------------------------------ */
expose u
if u~lastError \= .nil then call errormessage u~lastError
use arg dummy,sizeinfo
                                 /* wait for last size event msg then resize */
if self~PeekDialogMessage~left(8) \= "OnResize" then u~resize(self,sizeinfo)
```

The DlgAreaU method will then automatically resize & place the following widgets in your dialog when the dialog frame is dragged or the minimize, maximise or restore buttons are pressed:

Table 11-1. DlgAreaU Automatic Resize Elements

BLACKFRAME	BLACKRECT	BITMAPBUTTON	BUTTON
CHECKBOX	COMBOBOX	ENTRYLINE	GRAYFRAME

GRAYRECT	GROUPBOX	LISTBOX	LISTCONTROL
PASSWORDLINE	PROGRESSBAR	RADIOBUTTON	SCROLLBAR
SLIDERCONTROL	TABCONTROL	TEXT	TREECONTROL
WHITEFRAME	WHITERECT		

The following widgets can be defined in a UserDialog, but cannot be handled by the DlgAreaU resize method, so should not be used.

Table 11-2. DlgAreaU Non-Automatic Resize Elements

BUTTONGROUP	CHECKBOXSTEM	CHECKGROUP	COMBOINPUT
FULLSEQ	INPUT	INPUTGROUP	INPUTSTEM
OKCANCEL...	RADIOGROUP	RADIOSTEM	

All of these widgets can be achieved using combinations of the permitted methods.

11.4. Possible Problems

The DlgAreaU resize method can create slightly over-size margins on the left & bottom of the Dialog. To correct for this the DlgAreaU class has a CorrectionFactor attribute set by default to 1.05 which in tests appears to neutralise this effect. If your dialogs have over (or under) sized margins, you may be able to correct this in your code by overiding the Correction Factor like this (for example):

```
U=.DlgAreaU~new(self)
U~CorrectionFactor=1.05
```

You will have to experiment to find the apropriate setting for this attribute.

11.5. Sample Code

```
/* DlgAreaDemo.Rex  --  Demonstrate DlgArea & DlgAreaU Classes  --  Feb 2006 */

MyDlg=.MyDialog~new
MyDlg~execute("ShowTop")
MyDlg~DeInstall

exit
::requires "OODWIN32.CLS"
/* ========================================================================== */
::class MyDialog Subclass UserDialog
/* ========================================================================== */
::method Init
/* -------------------------------------------------------------------------- */
  self~Init:super
  rc=self~CreateCenter(250,250,"MyDialog","ThickFrame", ,"MS Sans Serif",8)
```

```
  self~InitCode=(rc=0)
  self~connectResize("OnResize")

/* ------------------------------------------------------------------------ */
::method DefineDialog
/* ------------------------------------------------------------------------ */
expose u

u=.dlgAreaU~new(self)                                        /* whole dlg   */
if u~lastError \= .nil then call errormessage u~lastError
e=.dlgArea~new(u~x       ,u~y       ,u~w("70%"),u~h("90%"))   /* edit   area */
s=.dlgArea~new(u~x       ,u~y("90%"),u~w("70%"),u~hr      )   /* status area */
b=.dlgArea~new(u~x("70%"),u~y       ,u~wr      ,u~hr      )   /* button area */

self~AddEntryLine(12,"text",e~x,e~y,e~w,e~h,"multiline")
self~AddText(s~x,s~y,s~w,s~h,"Status info appears here", ,11)

self~addButton(13,b~x,b~y("00%"),b~w,b~h("9%"),"Button" 0,"Button"||0)
self~addButton(14,b~x,b~y("10%"),b~w,b~h("9%"),"Button" 1,"Button"||1)
self~addButton(15,b~x,b~y("20%"),b~w,b~h("9%"),"Button" 2,"Button"||2)
self~addButton(16,b~x,b~y("30%"),b~w,b~h("9%"),"Button" 3,"Button"||3)
self~addButton(17,b~x,b~y("40%"),b~w,b~h("9%"),"Button" 4,"Button"||4)
self~addButton(18,b~x,b~y("50%"),b~w,b~h("9%"),"Button" 5,"Button"||5)
self~addButton(19,b~x,b~y("60%"),b~w,b~h("9%"),"Button" 6,"Button"||6)
self~AddButton( 1,b~x,b~y("90%"),b~w,b~h("9%"),"Ok","Ok","DEFAULT")

/* ------------------------------------------------------------------------ */
::method Unknown
/* ------------------------------------------------------------------------ */
use arg msgname, args
StatusLine = Self~GetStaticControl(11)
StatusLine~SetTitle("You Pressed" msgname)

/* ------------------------------------------------------------------------ */
::method OnResize
/* ------------------------------------------------------------------------ */
expose u
use arg dummy,sizeinfo
                                /* wait for last size event msg then resize */
if self~PeekDialogMessage~left(8) \= "OnResize" then u~resize(self,sizeinfo)
```

This achieves the same dialog as the previous DlgArea example, but now it is resizable by dragging the frame.

Chapter 12. AnimatedButton Class

The AnimatedButton class provides the methods to implement an animated button within a dialog. The attributes and methods are only described briefly in this document. An example program, `oowalker.rex`, is provided with the OODialog sample programs.

ParentDlg

Attribute holding the handle of the parent dialog

Stopped

Animation ends when set to 1 (see Stop method)

Init

Initialize the animation parameters:

```
but = .AnimatedButton~new(buttonid,from,to, ,
                          movex,movey,sizex,sizey,delay, ,
                          startx,starty,parentdialog)
```

The values are stored in a stem variable:

sprite.buttonid

ID of animation button

sprite.from

Array of in-memory bitmap handles, or a bitmap resource ID in a DLL, or the name of an array in the .local directory containing handles to bitmaps loaded with LoadBitmap. The array has to start with 1 and continue in increments by 1.

sprite.to

0 if sprite.from is an array, or the name of an array stored in `.local`, or a bitmap resource ID in a DLL

sprite.movex

Size of one move horizontally (pixels)

sprite.movey

Size of one move vertically

sprite.sizex

Horizontal size of all bitmaps (pixels)

sprite.sizey

Vertical size of all bitmaps

sprite.delay

Time delay between moves (ms)

Startx and starty are the initial bitmap position, and parentdialog is stored in the ParentDlg attribute.

Two more values are initialized in the stem variable:

sprite.smooth

Set to 1 for smooth edge change (can be changed to 0 for a bouncy edge change)

sprite.step

Set to 1 as the step size between sprite.from and sprite.to for bitmaps in a DLL

SetSprite

Set all the sprite. animation values using a stem:

```
mysprite.from = .array~of(bmp1,bmp2,...)
mysprite.to = 0
mysprite.movex = ...
...
self~setSprite(mysprite.)
```

GetSprite

Retrieve the animation values into a stem:

```
self~getSprite(mysprite.)
```

SetFromTo

Set bitmap information (sprite.from and sprite.to):

```
self~setFromTo(bmpfrom,bmpto)
```

SetMove

Set size of one move (sprite.movex and sprite.movey):

```
self~setMove(movex,movey)
```

SetDelay

Set delay between moves in milliseconds (sprite.delay):

```
self~setDelay(delay)
```

SetSmooth

Set smooth (1) or bouncy (0) edges (sprite.smooth):

```
self~setSmooth(smooth)  /* 1 or 0 */
```

SetStep

Set the step size (sprite.step) between sprite.from and sprite.to for bitmaps in a DLL, for example, if bitmap resources are numbered 202, 204, 206, etc:

```
self~setFromTo(202,210)
self~setStep(2)
```

Run

Run the animation by going through all the bitmaps repetitively until dialog is stopped; invokes MoveSeq:

```
self~run
```

MoveSeq

Animate one sequence through all the bitmaps in the given move steps; invokes MovePos:

```
self~moveSeq
```

MovePos

Move the bitmaps by the arguments:

```
self~movePos(movex,movey)
```

MoveTo

Move the bitmaps in the predefined steps to the given position; invokes MoveSeq:

```
self~moveTo(posx,posy)
```

SetPos

Set the new starting position of the bitmaps:

```
self~setPos(newx,newy)
```

GetPos

Retrieve the current position into a stem:

```
self~getPos(pos.)
say "pos=" pos.x pos.y
```

ParentStopped

Check the parent dialog window and return its finished attribute (1 means finished)

Stop

Stop animation by setting the stopped attribute to 1

HitRight

Invoked by run when the bitmap hits the right edge (returns 1 and bitmap starts at left again; you can return 0 and set the new position yourself)

HitLeft

Invoked when the bitmap hits the left edge (default action is to start at right again)

HitBottom

Invoked when the bitmap hits the bottom edge (default action is to start at top again)

HitTop

Invoked when the bitmap hits the top edge (default action is to start at bottom again)

To use an animated button a dialog has to:

- Define a button in a resource file (owner-drawn)
- Load the bitmaps of the animation into memory using an array
- Initialize the animated button with the animation parameters
- Invoke the run method of the animated button
- Stop the animation and remove the bitmaps from memory

The dialog may also dynamically change the parameters (for example, the size of a move, or the speed) and override actions, such as hitting an edge.

See the `oowalker.rex` and `oowalk2.rex` examples in `OODIALOG\SAMPLES`.

For further information see ConnectAnimatedButton.

Chapter 13. MessageExtensions Class

To use the methods defined by the mixin class MessageExtensions you must inherit from this class by specifying the INHERIT option for the ::CLASS directive in the class declaration. For example:

```
::class MyExtendedDialog SUBCLASS UserDialog INHERIT MessageExtensions
```

Requires:

The MessageExtensions class requires the class definition file `oodwin32.cls`:

```
::requires oodwin32.cls
```

Methods:

Instances of the MessageExtensions class implement the methods listed in MessageExtensions Instance Methods table.

Table 13-1. MessageExtensions Instance Methods

Method...	...on page
ConnectButtonNotify	ConnectButtonNotify
ConnectComboBoxNotify	ConnectComboBoxNotify
ConnectCommonNotify	ConnectCommonNotify
ConnectEditNotify	ConnectEditNotify
ConnectListBoxify	ConnectListBoxNotify
ConnectListNotify	ConnectListNotify
ConnectListViewNotify	ConnectListViewNotify
ConnectScrollBarNotify	ConnectScrollBarNotify
ConnectSliderNotify	ConnectScrollBarNotify
ConnectTabNotify	ConnectTabNotify
ConnectTreeNotify	ConnectTreeNotify
DefListDragHandler	DefListDragHandler
DefTreeDragHandler	DefTreeDragHandler

13.1. ConnectCommonNotify

```
>>-aMessageExtensions~ConnectCommonNotify(--id--,--"--+-OUTOFMEMORY-+-->
                                                     +-CLICK-------+
                                                     +-DBLCLK------+
                                                     +-ENTER-------+
                                                     +-RCLICK------+
                                                     +-RDBLCLK-----+
                                                     +-GOTFOCUS----+
                                                     +-LOSTFOCUS---+
```

```
>--"--+---------------+--)------------------------------------><
      +-,--msgToRaise-+
```

The ConnectCommonNotify method connects a particular WM_NOTIFY message for a dialog control with a method. The WM_NOTIFY message informs the dialog that an event has occurred with regard to a dialog control.

If a specific Connect...Notify method exists for the dialog control, use this method instead of the ConnectCommonNotify method because the system might send a specific notification instead of a common one.

Arguments:

The arguments are:

id

The ID of the dialog control of which a notification is to be connected to a method.

event

The event to be connected with a method:

OUTOFMEMORY

The dialog control went out of memory.

CLICK

The left mouse button was clicked on the dialog control.

DBLCLK

The left mouse button was double-clicked on the dialog control.

ENTER

The return key was pressed in the dialog item.

RCLICK

The right mouse button was clicked on the dialog item.

RDBLCLK

The right mouse button was double-clicked on the dialog control.

GOTFOCUS

The dialog item got the input focus.

LOSTFOCUS

The dialog item lost the input focus.

msgToRaise

The message that is to be sent whenever the specified notification is received. Provide a method with a matching name. If you omit this argument, the event is preceded by `On`.

Return value:

The return codes are:

0

No error detected.

-1

The resource ID could not be resolved or the event argument is incorrect.

1

The message was not connected correctly.

Example:

The following example connects the double-click of the left mouse button on dialog control DLGITEM1 with method OnDblClk:

```
::class MyDlgClass subclass UserDialog inherit MessageExtensions

::method Init
  self~init:super(...)
  self~ConnectCommonNotify(DLGITEM1, "DBLCLK")

::method OnDblClk
  use arg id
  say "Item" id " has been double-clicked!"
```

Note: Connections are usually placed in the Init or InitDialog method. If both methods are defined, use init as the place for this connection - but not before init:super has been called.

13.2. ConnectTreeNotify

```
>>-aMessageExtensions~ConnectTreeNotify(--id--,--"--+-BEGINDRAG---+--"-->
                                                    +-BEGINRDRAG--+
                                                    +-BEGINEDIT---+
                                                    +-ENDEDIT-----+
                                                    +-DEFAULTEDIT-+
                                                    +-EXPANDING---+
                                                    +-EXPANDED----+
                                                    +-DELETE------+
                                                    +-KEYDOWN-----+
                                                    +-SELCHANGING-+
```

```
                                                        +-SELCHANGED--+

>--+---------------+--)---------------------------------------><
   +-,--msgToRaise-+
```

The ConnectTreeNotify method connects a particular WM_NOTIFY message for a tree view control with a method. The WM_NOTIFY message informs the dialog that an event has occurred in the tree view.

Arguments:

The arguments are:

id

The ID of the tree view control of which a notification is to be connected to a method.

event

The event to be connected with a method:

BEGINDRAG

A drag-and-drop operaton was initiated. See DefTreeDragHandler for information on how to implement a drag-and-drop handler.

BEGINRDRAG

A drag-and-drop operaton involving the right mouse button was initiated. See DefTreeDragHandler for information on how to implement a drag-and-drop handler.

BEGINEDIT

Editing a label has been started.

ENDEDIT

Label editing has ended.

DEFAULTEDIT

This event connects the notification that label editing has been started and ended with a predefined event-handling method. This method extracts the newly entered text from the notification and modifies the item of which the label was edited. If this event is not connected you must provide your own event-handling method and connect it with the BEGINEDIT and ENDEDIT events. Otherwise, the edited text is lost and the item remains unchanged.

When you specify this event, omit the *msgToRaise* argument.

EXPANDING

An item is about to expand or collapse. This notification is sent before the item has expanded or collapsed.

EXPANDED

An item has expanded or collapsed. This notification is sent after the item expanded or collapsed.

DELETE

An item has been deleted.

KEYDOWN

A key was pressed inside the tree view. This notification is not sent while a label is being edited.

SELCHANGING

Another item is about to be selected. This notification is sent before the selection has changed.

SELCHANGED

Another item was selected. This notification is sent after the selection was changed.

msgToRaise

The message that is to be sent whenever the specified notification is received from the tree view control. Provide a method with a matching name. If you omit this argument, the event is preceded by `On`.

Return value:

The return codes are:

0

No error detected.

-1

The resource ID could not be resolved or the event argument is incorrect.

1

The message was not connected correctly.

Example:

The following example connects the selection-changed event for the tree view FileTree with method NewTreeSelection and displays the text of the new selection:

```
::class MyDlgClass subclass UserDialog inherit MessageExtensions

::method Init
  self~init:super(...)
```

```
  self~ConnectTreeNotify("FileTree", "SELCHANGED", "NewTreeSelection")

::method NewTreeSelection
  tc = self~GetTreeControl("FileTree")
  info. = tc~ItemInfo(tc~Selected)
  say "New selection is:" info.!text
```

Note:

1. Connections are usually placed in the Init or InitDialog method. If both methods are defined, use init as the place for this connection - but not before init:super has been called.
2. The event-handling method connected to ENDEDIT receives two arguments: the item handle of which the label has been edited and the newly entered text. Example:

   ```
   ::method OnEndEdit
     use arg item, newText
   ```

3. The event-handling method connected to KEYDOWN receives two arguments: the control ID of the tree view control and the virtual key code pressed. Use the method KeyName of the VirtualKeyCodes class to get the name of the key. Note that your class must inherit from the VirtualKeyCodes class to use the KeyName method. Example:

   ```
   ::method OnKeyDown
     use arg id, vkey
     say "Key" self~KeyName(vkey) "was pressed."
   ```

4. The event-handling method connected to EXPANDED or EXPANDING receives three arguments: the control ID of the tree view control, the tree item expanded or collapsed, and a string that indicates whether the item was expanded or collapsed. Example:

   ```
   ::method OnExpanding
     use arg id, item, what
     say "Item with handle" item "is going to be" what
   ```

5. The event-handling method connected to BEGINDRAG or BEGINRDRAG receives three arguments: the control ID of the tree view control, the tree item to be dragged, and the point where the mouse cursor was pressed (x and y positions, separated by a blank). Example:

   ```
   ::method OnBeginDrag
     use arg id, item, where
     say "Item with handle" item "is in drag-and-drop mode"
     parse var where x y
     say "The drag operation started at point ("x","y")"
   ```

13.3. DefTreeDragHandler

```
>>-aMessageExtensions~DefTreeDragHandler----------------------><
```

A tree view control cannot handle a drag-and-drop operation within the tree view. Therefore, you can connect the DefTreeDragHandler method with the BEGINDRAG or BEGINRDRAG notification message (see ConnectTreeNotify) to allow the moving of an item or a node with all its subitems from

one parent node to another within a tree view. The cursor shape is changed to a crosshair during the drag operation. If the cursor is moved over the item dragged, the cursor shape is changed to a slashed circle. You can cancel the drag operation by clicking the other mouse button while holding the button that started the drag operation.

The DefTreeDragHandler is implemented as follows:

```
::method DefTreeDragHandler
   use arg id, item, pt
   tc = self~GetTreeControl(id)
   hc = tc~Cursor_Cross   /* change cursor and store current */
   parse value tc~GetRect with left top right bottom
   oldItem = 0
   nocurs = 0
   lmb = self~IsMouseButtonDown("LEFT")
   rmb = self~IsMouseButtonDown("RIGHT")
   call time "R"
   do while (lmb \= 0 | rmb \= 0) & \(lmb \= 0 & rmb \= 0)
     pos = self~CursorPos
     parse var pos x y
     parse value tc~ScreenToClient(x, y) with newx newy
      ht = tc~HitTest(newx, newy)
      if ht \= 0 & ht~wordpos("ONITEM") > 0 then do
          parse var ht newParent where
          /* check if droptarget is the current parent or one of the dragged
             item&apos;s children */
          if newParent \= Item & newParent \= tc~Parent(Item) & tc~IsAncestor,
             (Item,newParent) = 0
          then do
             is. = tc~ItemInfo(newParent)
             if is.!State~Wordpos("INDROP") = 0 then
             do
                call time "R"
                tc~DropHighlight(newParent)
                if nocurs \= 0 then do
                  tc~RestoreCursorShape(nocurs) /*restore old cursor (cross)*/
                  nocurs = 0
                end
             end
             else if time("E") > 1 then do  /* expand node after 1 second */
                 if is.!Children \= 0 & is.!State~Wordpos("EXPANDED") = 0 then
                 tc~expand(newParent)
             end
          end
          else do
             if nocurs = 0 then do
                nocurs = tc~Cursor_No  /* set no cursor and retrieve
                                          current cursor (cross) */
                tc~DropHighlight  /* remove drop highlight */
             end
          end
      end
```

```
      else do
          if newParent \= 0 then do
             /* necessary to redraw cursor when moving on a valid item again */
              tc~DropHighlight  /* remove drop highlight */
              newParent = 0
          end
          if nocurs = 0 then nocurs = tc~Cursor_No  /* set no cursor and
                                      retrieve current cursor (cross) */
          /* handle scrolling */
          fvItem = tc~FirstVisible
          if (ybottom) & (tc~NextVisible(fvItem) \= 0) then do
              tc~MakeFirstVisible(tc~NextVisible(fvItem))
              if y-bottom < 200 then call sleepms 200-(y-bottom)
          end
      end
      lmb = self~IsMouseButtonDown("LEFT")
      rmb = self~IsMouseButtonDown("RIGHT")
   end
   if ht~wordpos("ONITEM") > 0 & lmb = 0 & rmb = 0 then do /* if mouse on item
                                                 and both mouse buttons up */
       item = tc~MoveItem(Item, newParent, 1)  /* move item under newParent */
   end
   tc~DropHighlight(0)  /* remove drop highlight */
   tc~select(item)      /* select item */
   tc~EnsureVisible(item)
   tc~RestoreCursorShape(hc)  /* restore old cursor */
   pos = self~CursorPos
   parse var pos x y
   self~SetCursorPos(x+1, y+1)  /* move cursor to force redraw */
```

13.4. ConnectListNotify

```
>>-aMessageExtensions~ConnectListNotify(--id--,--"--+-ACTIVATE----+--"-->
                                                    +-BEGINDRAG---+
                                                    +-BEGINRDRAG--+
                                                    +-BEGINEDIT---+
                                                    +-ENDEDIT-----+
                                                    +-DEFAULTEDIT-+
                                                    +-CHANGING----+
                                                    +-CHANGED-----+
                                                    +-COLUMNCLICK-+
                                                    +-DELETE------+
                                                    +-DELETEALL---+
                                                    +-INSERTED----+
                                                    +-KEYDOWN-----+

>--+---------------+--)---------------------------------------><
   +-,--msgToRaise-+
```

The ConnectListNotify method connects a particular WM_NOTIFY message for a list view control with a method. The WM_NOTIFY message informs the dialog that an event has occurred in the list view.

Arguments:

The arguments are:

id

The ID of the list view control of which a notification is to be connected to a method.

event

The event to be connected with a method:

ACTIVATE

An item is activated by double-clicking the left mouse button.

BEGINDRAG

A drag-and-drop operaton was initiated. See DefListDragHandler for information on how to implement a drag-and-drop handler.

BEGINRDRAG

A drag-and-drop operaton involving the right mouse button was initiated. See DefListDragHandler for information on how to implement a drag-and-drop handler.

BEGINEDIT

Editing a label has been started.

ENDEDIT

Label editing has ended.

DEFAULTEDIT

This event connects the notification that label editing has been started and ended with a predefined event-handling method. This method extracts the newly entered text from the notification and modifies the item of which the label was edited. If this event is not connected you must provide your own event-handling method and connect it with the BEGINEDIT and ENDEDIT events. Otherwise, the edited text is lost and the item remains unchanged.

When you specify this event, omit the *msgToRaise* argument.

CHANGING

An item is about to change. This notification is sent before the item is changed.

CHANGED

An item has changed. This notification is sent after the item changed.

COLUMNCLICK

A column has been clicked.

DELETE

An item has been deleted.

DELETEALL

All items have been deleted.

INSERTED

A new item has been inserted.

KEYDOWN

A key was pressed inside the list view. This notification is not sent while a label is being edited.

msgToRaise

The message that is to be sent whenever the specified notification is received from the list view control. Provide a method with a matching name. If you omit this argument, the event is preceded by `On`.

Return value:

The return codes are:

0

No error detected.

-1

The resource ID could not be resolved or the event argument is incorrect.

1

The message was not connected correctly.

Example:

The following example connects the column-clicked event for the list view EMPLOYEES with method ColumnAction and changes the style of the list view from REPORT to SMALLICON:

```
::class MyDlgClass subclass UserDialog inherit MessageExtensions

::method Init
  self~init:super(...)
  self~ConnectListNotify("EMPLOYEES", "COLUMNCLICK", "ColumnAction")

::method ColumnAction
  use arg id, column
  lc = self~GetListControl("EMPLOYEES")
```

```
lc~ReplaceStyle("REPORT", "SMALLICON EDIT SINGLESEL ASCENDING")
if column > 0 then ...
```

1. Connections are usually placed in the Init or InitDialog method. If both methods are defined, use init as the place for this connection - but not before init:super has been called.
2. The event-handling method connected to ENDEDIT receives two arguments: the item ID of which the label has been edited and the newly entered text.

   Example:

   ```
   ::method OnEndEdit
     use arg item, newText
   ```

3. The event-handling method connected to COLUMNCLICK receives two arguments: the control ID of the list view control and the zero-based column number of which the header button was pressed.

   Example:

   ```
   ::method OnColumnClick
     use arg id, column
   ```

4. The event-handling method connected to KEYDOWN receives two arguments: the control ID of the list view control and the virtual key code pressed. Use the method KeyName of the VirtualKeyCodes class to get the name of the key. Note that your class must inherit from the VirtualKeyCodes class to use the KeyName method.

   Example:

   ```
   ::method OnKeyDown
     use arg id, vkey
     say "Key" self~KeyName(vkey) "was pressed."
   ```

5. The event-handling method connected to BEGINDRAG or BEGINRDRAG receives three arguments: the control ID of the list view control, the index of the list item to be dragged, and the point where the mouse cursor was pressed (x and y positions, separated by a blank).

   Example:

   ```
   ::method OnBeginDrag
     use arg id, item, where
     say "Item at index" item "is in drag-and-drop mode"
     parse var where x y
     say "The drag operation started at point ("x","y")"
   ```

13.5. DefListDragHandler

```
>>-aMessageExtensions~DefListDragHandler----------------------><
```

A list view control cannot handle a drag-and-drop operation within the list view. Therefore, you can connect the DefListDragHandler method with the BEGINDRAG or BEGINRDRAG notification message (see ConnectListNotify) to allow the dragging of an item from one location to another within an icon view and a smallicon view. The cursor shape is changed to a crosshair during the drag operation.

You can cancel the drag operation by clicking the other mouse button while holding the button that started the drag operation. Note that the item position is not flexible when the list view control has the AUTOARRANGE style.

The DefListDragHandler is implemented as follows:

```
::method DefListDragHandler
   use arg id, item, pt
   lc = self~GetListControl(id)
   hc = lc~Cursor_Cross   /* change cursor and store current */
   parse value lc~GetRect with left top right bottom
   parse var pt oldx oldy
   origin = lc~ItemPos(item)
   lmb = self~IsMouseButtonDown("LEFT")
   rmb = self~IsMouseButtonDown("RIGHT")
   do while (lmb \= 0 | rmb \= 0) & \(lmb \= 0 & rmb \= 0)
     pos = self~CursorPos
     parse var pos x y
     parse value lc~ScreenToClient(x, y) with newx newy
     hs = lc~HScrollPos; vs = lc~VScrollPos
     sx = x-right
     sy = y-bottom
      in_rx = (sx <= 30) & (newx >= -30)
      in_ry = (sy <= 30) & (newy >= -30)
      if (in_rx & in_ry) then do    /* is the mouse cursor inside the drag
                                                        rectangle */
          if xright then sx = sx + 30; else sx = 0
          if ybottom then sy = sy + 30; else sy = 0
          newx = newx+hs;  newy = newy +vs;
          if newx < 0 then newx = 0
          if newy < 0 then newy = 0
          if (in_rx & oldx \= newx) | (in_ry & oldy \= newy) then do
           lc~SetItemPos(item, newx, newy)
             oldx = newx
             oldy = newy
             if sx \= 0 | sy \= 0 then do
                lc~Scroll(sx, sy)
                call sleepms 30
             end
        end
     end
     else do    /* no, so force the mouse cursor back inside the rectangle */
        if newx < -30 then newx = -30
        if sx > 30 then newx = (right-left) + 28
        if newy < -30 then newy = -30
        if sy > 30 then newy = (bottom-top) + 28
        parse value lc~ClientToSCreen(newx, newy) with x y
        self~SetCursorPos(x, y)
     end
     lmb = self~IsMouseButtonDown("LEFT")
     rmb = self~IsMouseButtonDown("RIGHT")
   end
   if (lmb \= 0 & rmb \= 0) then do  /* if both buttons pressed restore
```

```
                                        original pos */
      parse var origin x y
      lc~SetItemPos(item, x, y)
   end
   lc~RestoreCursorShape(hc)  /* restore old cursor */
   pos = self~CursorPos
   parse var pos x y
   self~SetCursorPos(x+1, y+1)  /* move cursor to force redraw */
```

13.6. ConnectListViewNotify

```
>>-aMessageExtensions~ConnectListViewNotify(--id--,--------------------->

>--"--+-CLICK--------------+--"--+---------------+--)-----------------><
      +-CHECKBOXCHANGED----+     +-,--methodName-+
      +-SELECTCHANGED------+
      +-FOCUSCHANGED-------+
      +-SELECTFOCUSCHANGED-+
```

Microsoft continually enhances the Windows User Interface and therefore the dialog controls evolve over time. The `ConnectListViewNotify` method is similar to the ConnectListNotify method, but it is designed to take advantage of new features in the list-view control. It connects notification message sent by a list view control to its parent dialog with a method, provided by the programmer, in an ooDialog class. These notifications inform the dialog that an event has occurred in the list view.

The `ConnectListViewNotify` method is designed to provide more information to the invoked method than the `ConnectListNotify` or `ConnectCommonNotify` methods. Connecting an event through the `ConnectListViewNotify` method is used as a replacement for connecting the same event through the `ConnectListNotify` method or the `ConnectCommonNotify` method.

> **Note:** If the same event, for the same control, is connected using two different ConnectXXX methods, only one connection will be in effect. This will be the connection whose ConnectXXX method is invoked first. For example, take a dialog that has a list view control with resource ID of 109. If the mouse click event is connected for that control using the `ConnectCommonNotify` method and then the mouse click event is also connected using the `ConnectListViewNotify` method, only one connection will be active. Which one is active is dependent on the order of invocation of the ConnectXXX methods.

Each of the events that can be connected through the `ConnectListViewNotify` method is a replacement for connecting that event through one of the other ConnectXXX methods. The argument documentation provides more detail on this.

Arguments:

The arguments are:

id

The ID of the list view control for which the event is to be connected to a method.

event

The event to be connected with a method:

Note: When using `ConnectListViewNotify` a separate method can be connected to each of the checkbox changed, selection changed, and focus changed events. These event connectionss are all replacements for the ConnectListNotify method's CHANGED event.

CLICK

An item was clicked with the left mouse button. Connecting the CLICK event is a replacement for the `ConnectCommonNotify` method's CLICK event.

CHECKBOXCHANGED

The check box state of an item changed. (The check box was checked or unchecked.) This event can only occur if the list view has the check box extended list view style.

SELECTCHANGED

The selection state of an item changed. (The item was selected or unselected.)

FOCUSCHANGED

The focus state of an item changed. (The item gained or lost the focus.)

SELECTFOCUSCHANGED

The selection state or the focus state of an item changed. This event argument combines the selection changed and the focus changed event into one connection. When this event is connected, separate selection changed and focus changed events can not be connected. This keyword can be abbreviated to "SELECTFOCUS".

methodName

The method that is to be invoked whenever the specified event occurs. Provide a method with a matching name in the dialog class. If you omit this argument, a default method name will be constructed. This name will be the name of the event preceded by `On`.

Return value:

The return codes are:

0

No error detected.

-1

The resource ID could not be resolved or the event argument is incorrect.

1

The event was not connected correctly to a method.

Event Methods

The following details the method signature for each of the `ConnectListViewNotify` events:

CLICK

The method invoked for the left mouse click event recieves four arguments.

```
::method onClick
use arg id, itemIndex, columnIndex, keyState
```

id

The resource ID of the list view control whose item was clicked.

itemIndex

The index of the item that was clicked.

columnIndex

The index of the subitem that was clicked. This will be the column clicked on if the list view is in report view. In other views it will be 0.

keyState

This argument reports the state of the shift, control, and alt keys at the time of the mouse click. The argument is a string of keywords separated by blanks. The keywords consist of: SHIFT, CONTROL, ALT, or NONE. The presence of a keyword indicates the key was held down when the user clicked on the list view control. NONE of course indicates that none of the keys were down. If the user managed to hold all three of the keys down at the time of the mouse click, the argument would be the string: "SHIFT CONTROL ALT"

CHECKBOXCHANGED

The method invoked for the checkbox changed event recieves three arguments. The method signature will look as follows:

```
::method onCheckboxChanged
  use arg id, itemIndex, state
```

id

The resource ID of the list view control whose item had the checkbox state changed.

itemIndex

The index of the item whose checkbox was changed.

state

This argument reports whether the check box was checked or unchecked. Its value will be either "CHECKED" or "UNCHECKED"

SELECTCHANGE

The method invoked when the selection changes will recieve three arguments. The method signature will look as follows:

```
::method onSelectChanged
  use arg id, itemIndex, state
```

id

The resource ID of the list view control whose item had the selection changed.

itemIndex

The index of the item whose selection was changed.

state

This argument reports whether the item was selected or unselected. Its value will be either "SELECTED" or "UNSELECTED"

FOCUSCHANGED

The method invoked for the focus changed event recieves three arguments. The method signature will look as follows:

```
::method onFocusChanged
  use arg id, itemIndex, state
```

id

The resource ID of the list view control whose item had the focus changed.

itemIndex

The index of the item which gained or lost the focus.

state

This argument reports whether the focus was gained or lost. Its value will be either "FOCUSED" or "UNFOCUSED"

SELECTFOCUSCHANGED

The method invoked for the selection or focus changed event recieves three arguments. The method signature will look as follows:

```
::method onSelectFocusChanged
  use arg id, itemIndex, state
```

id

The resource ID of the list view control whose item had the either the focus or the selection changed.

itemIndex

The index of the item where the state was changed.

state

This argument reports whether the focus was gained or lost and whether the selection was gained or lost. Its value will contain at least one of the keywords: "SELECTED", "UNSELECTED", "FOCUSED" or "UNFOCUSED". It is possible for both the selection and focus changed to be reported at once, however sometimes each change is reported separately. (This has nothing to do with ooDialog, it is how the operating system sends the messages.)

Example:

The following example is from an address book application. A list view control is filled with the information from the address book, one item for each entry. The check box changed event is connected to the onCheckboxChanged method. The onCheckboxChanged method will receive 3 arguments: the resource ID of the control, the index of the item whose check box changed, and the changed state. If the user checks the check box, that entry is added to a mail merge being constructed. If the user unchecks the box, the entry is removed from the mail merge.

```
::class MailingListDlg subclass UserDialog inherit AdvancedControls MessageExtensions

::method initDialog
  expose mailList

  ...
  mailList = self~getListControl(IDC_LV_ADDRESSES)
  ...

  -- Since the methodName argument is omitted, ooDialog will construct a default
  -- name of 'onCheckboxChanged'
  self~connectListViewNotify(IDC_LV_ADDRESSES, "CHECKBOXCHANGED")
  ...

::method onCheckboxChanged
  expose mailList
  use arg id, itemIndex, state

  if state == "CHECKED" then
```

```
      self~addToMailMerge(mailList, itemIndex)
    else
      self~removeFromMailMerge(mailList, itemIndex)
```

13.7. ConnectButtonNotify

```
>>-aMessageExtensions~ConnectButtonNotify(--id--,--"--+-CLICKED---+--"-->
                                                      +-DBLCLK----+
                                                      +-DISABLE---+
                                                      +-GOTFOCUS--+
                                                      +-LOSTFOCUS-+
                                                      +-HILITE----+
                                                      +-UNHILITE--+
                                                      +-PAINT-----+

>--+---------------+--)---------------------------------------><
   +-,--msgToRaise-+
```

The ConnectButtonNotify method connects a particular WM_NOTIFY message for a button control (push button, radio button, or check box) with a method. The WM_NOTIFY message informs the dialog that an event has occurred with regard to the button.

> **Note:** In order to recieve the GOTFOCUS, LOSTFOCUS, and DBLCLK events, the button control has to have the NOTIFY (BS_NOTIFY) style. For user defined dialogs use the NOTIFY style keyword in the Add... method when the button is defined. For dialogs created from a compiled resource or a resource script file use the BS_NOTIFY style for the button resource. The other events are always sent and it is not necessary to add the NOTIFY style.

Arguments:

The arguments are:

id

The ID of the button control of which a notification is to be connected to a method.

event

The event to be connected with a method:

CLICKED

The button has been clicked.

DBLCLK

The button has been double-clicked.

DISABLE

The button has been disabled.

GOTFOCUS

The button got the input focus.

LOSTFOCUS

The button lost the input focus.

HILITE

The button has been selected.

UNHILITE

The highlighting is to be removed (lost selection).

PAINT

The button is to be repainted. This notification is only sent for owner-drawn buttons.

msgToRaise

The message that is to be sent whenever the specified notification is received from the button control. Provide a method with a matching name. If you omit this argument, the event is preceded by `On`.

Return value:

The return codes are:

0

No error detected.

-1

The resource ID could not be resolved or the event argument is incorrect.

1

The message was not connected correctly.

Example:

The following example displays a message whenever the OK button is selected:

```
::class MyDlgClass subclass UserDialog inherit MessageExtensions

::method Init
  self~init:super(...)
  self~ConnectButtonNotify("OK", "HILITE")

::method OnHilite
  say "The OK button has been selected"
```

1. Connections are usually placed in the Init or InitDialog method. If both methods are defined, use init as the place for this connection - but not before init:super has been called.
2. The event-handling methods receive two arguments: the ID of the button (extract the low-order word) and the handle to the button. Example:

```
::method Handler
  use arg ev_id, handle
  id = BinaryAnd(ev_id, "0x0000FFFF")
  ...
```

13.8. ConnectEditNotify

```
>>-aMessageExtensions~ConnectEditNotify(--id--,--"--+-CHANGE----+--"-->
                                                    +-UPDATE----+
                                                    +-ERRSPACE--+
                                                    +-MAXTEXT---+
                                                    +-HSCROLL---+
                                                    +-VSCROLL---+
                                                    +-GOTFOCUS--+
                                                    +-LOSTFOCUS-+

>--+---------------+--)-------------------------------------------><
   +-,--msgToRaise-+
```

The ConnectEditNotify method connects a particular WM_NOTIFY message for an edit control with a method. The WM_NOTIFY message informs the dialog that an event has occurred with regard to the edit control.

Arguments:

The arguments are:

id

The ID of the edit control of which a notification is to be connected to a method.

event

The event to be connected with a method:

CHANGE

The text has been altered. This notification is sent after the screen has been updated.

UPDATE

The text has been altered. This notification is sent before the screen is updated.

ERRSPACE

An out-of-memory problem has occurred.

MAXTEXT

The text inserted exceeds the specified number of characters for the edit control. This notification is also sent when:

- An edit control does not have the ES_AUTOHSCROLL or AUTOSCROLLH style and the number of characters to be inserted would exceed the width of the edit control.
- The ES_AUTOVSCROLL or AUTOSCROLLV style is not set and the total number of lines resulting from a text insertion would exceed the height of the edit control.

HSCROLL

The horizontal scroll bar has been used.

VSCROLL

The vertical scroll bar has been used.

GOTFOCUS

The edit control got the input focus.

LOSTFOCUS

The edit control lost the input focus.

msgToRaise

The message that is to be sent whenever the specified notification is received from the edit control. Provide a method with a matching name. If you omit this argument, the event is preceded by `On`.

Return value:

The return codes are:

0

No error detected.

-1

The resource ID could not be resolved or the event argument is incorrect.

1

The message was not connected correctly.

Example:

The following example verifies the input of entry line AMOUNT and resets it to 0 when a nonnumeric value was entered:

```
::class MyDlgClass subclass UserDialog inherit MessageExtensions

::method Init
  self~init:super(...)
  self~ConnectEditNotify("AMOUNT", "CHANGE")

::method OnChange
  ec = self~GetEditControl("AMOUNT")
  if ec~GetText~Space(0) \= "" & ec~GetText~DataType("N") = 0 then do
    ec~SetModified(0)
    ec~Select
    ec~ReplaceSelText("0")
   end
```

1. Connections are usually placed in the Init or InitDialog method. If both methods are defined, use init as the place for this connection - but not before init:super has been called.
2. The event-handling methods receive two arguments: the ID of the edit control (extract the low-order word) and the handle to the edit control.

   Example:

   ```
   ::method Handler
     use arg ev_id, handle
     id = BinaryAnd(ev_id, "0x0000FFFF")
     ...
   ```

13.9. ConnectListBoxNotify

```
>>-aMessageExtensions~ConnectListBoxNotify(--id--,--"--+-DBLCLK----+--"-->
                                                      +-ERRSPACE--+
                                                      +-GOTFOCUS--+
                                                      +-LOSTFOCUS-+
                                                      +-SELCANCEL-+
                                                      +-SELCHANGE-+

>--+---------------+--)-----------------------------------------------><
   +-,--msgToRaise-+
```

The ConnectListBoxNotify method connects a particular WM_NOTIFY message for a list box with a method. The WM_NOTIFY message informs the dialog that an event has occurred in the list box.

Arguments:

The arguments are:

id

The ID of the list box of which a notification is to be connected to a method.

event

The event to be connected with a method:

DBLCLK

An entry in the list box has been selected with a double click.

ERRSPACE

An out-of-memory problem has occurred.

GOTFOCUS

The list box got the input focus.

LOSTFOCUS

The list box lost the input focus.

SELCANCEL

The selection in the list box has been canceled.

SELCHANGE

Another list box entry has been selected.

msgToRaise

The message that is to be sent whenever the specified notification is received from the list box. Provide a method with a matching name. If you omit this argument, the event is preceded by `On`.

Return value:

The return codes are:

0

No error detected.

-1

The resource ID could not be resolved or the event argument is incorrect.

1

The message was not connected correctly.

Example:

The following example displays the text of the selected list box entry:

```
::class MyDlgClass subclass UserDialog inherit MessageExtensions

::method Init
  self~init:super(...)
  self~ConnectListBoxNotify("MYLIST", "SELCHANGE", "SelectionChanged")

::method SelectionChanged
  li = self~GetListBox("MYLIST")
  say "New selection is:" li~Selected
```

Notes:

1. Connections are usually placed in the Init or InitDialog method. If both methods are defined, use init as the place for this connection - but not before init:super has been called.
2. The event-handling methods receive two arguments: the ID of the list box (extract the low-order word) and the handle to the list box. Example:

```
::method Handler
  use arg ev_id, handle
  id = BinaryAnd(ev_id, "0x0000FFFF")
  ...
```

13.10. ConnectComboBoxNotify

```
>>-aMessageExtensions~ConnectComboBoxNotify(--id--,--"--+-CHANGE-------+--"-->
                                                       +-UPDATE-------+
                                                       +-CLOSEUP------+
                                                       +-DROPDOWN-----+
                                                       +-DBLCLK-------+
                                                       +-ERRSPACE-----+
                                                       +-GOTFOCUS-----+
                                                       +-LOSTFOCUS----+
                                                       +-SELCHANGE----+
                                                       +-SELENDOK-----+
                                                       +-SELENDCANCEL-+

>--+---------------+--)-----------------------------------------------------><
   +-,--msgToRaise-+
```

The ConnectComboBoxNotify method connects a particular WM_NOTIFY message for a combo box with a method. The WM_NOTIFY message informs the dialog that an event has occurred in the combo box.

Arguments:

The arguments are:

id

The resource ID of the combo box for which a notification is to be connected to a method.

event

The event to be connected with a method:

CHANGE

The text in the edit control has been altered. This notification is sent after Windows updated the screen.

UPDATE

The text in the edit control has been altered. This notification is sent before Windows updates the screen.

CLOSEUP

The list of the combo box has been closed.

DROPDOWN

The list of the combo box is about to be made visible.

DBLCLK

An entry in the combo box list has been selected with a double click.

ERRSPACE

An out-of-memory problem has occurred.

GOTFOCUS

The combo box got the input focus.

LOSTFOCUS

The combo box lost the input focus.

SELCHANGE

Another entry in the combo box list has been selected.

SELENDOK

The list was closed after another entry was selected.

SELENDCANCEL

After the selection of another entry, another control or dialog was selected, which canceled the selection of the entry.

msgToRaise

The message that is to be sent whenever the specified notification is received from the combo control. Provide a method with a matching name. If you omit this argument, the event is preceded by `On`.

Return value:

The return codes are:

-1

The resource ID could not be resolved or the event argument is incorrect.

0

No errors were detected.

1

The message was not connected correctly.

Example:

The following example invokes method PlaySong whenever the list of the combo box with the resource ID of PROFESSIONS is about to be made visible. In this case PROFESSIONS is a symbolic ID that has been added to the ConstDir directory of the MyDlgClass in another part of the program:

```
::class MyDlgClass subclass UserDialog inherit MessageExtensions

::method InitDialog
  self~init:super(...)
  self~ConnectComboBoxNotify("PROFESSIONS", "DROPDOWN", "PlaySong")
```

Notes:

1. Connections are usually placed in the Init or InitDialog method. If both methods are defined, use init as the place for this connection - but not before init:super has been called.
2. The event-handling methods receive two arguments: the first is a combination of the ID of the combo box and the ID of the notification messages. (Extract the low-order word to get the combo box ID.) The second argument is the window handle of the combo box. Example:

```
::method PlaySong
  use arg eventID, handle
  id = BinaryAnd(eventID, "0x0000FFFF")
  if id == self~ConstDir["PROFESSIONS"] then
    -- take some action
  ...
```

13.11. ConnectScrollBarNotify

```
>>-aMessageExtensions~ConnectScrollBarNotify(--id--,--"--+-UP--------+--"-->
                                                        +-DOWN------+
                                                        +-TOP-------+
                                                        +-BOTTOM----+
                                                        +-PAGEUP----+
                                                        +-PAGEDOWN--+
                                                        +-DRAG------+
                                                        +-ENDSCROLL-+
                                                        +-POSITION--+

>--+---------------+--)-----------------------------------------------><
   +-,--msgToRaise-+
```

The ConnectScrollBarNotify method connects a particular WM_NOTIFY message for a scroll bar with a method. The WM_NOTIFY message informs the dialog that an event has occurred with regard to the scroll bar.

Arguments:

The arguments are:

id

The ID of the scroll bar of which a notification is to be connected to a method.

event

The event to be connected with a method:

UP

The scroll bar was scrolled to the left or up by one unit.

DOWN

The scroll bar was scrolled to the right or down by one unit.

TOP

The scroll bar was scrolled to the upper left.

BOTTOM

The scroll bar was scrolled to the lower right.

PAGEUP

The scroll bar was scrolled to the left or up by one page size.

PAGEDOWN

The scroll bar was scrolled to the right or down by one page size.

DRAG

The scroll bar has been dragged.

ENDSCROLL

Scrolling has been ended, that is, the appropriate key or mouse button has been released.

POSITION

The scroll bar was scrolled to an absolute position (the left mouse button has been released).

msgToRaise

The message that is to be sent whenever the specified notification is received from the scroll bar. Provide a method with a matching name. If you omit this argument, the event is preceded by `On`.

Return value:

The return codes are:

0

No error detected.

-1

The resource ID could not be resolved or the event argument is incorrect.

1

The messages was not connected correctly.

Example:

The following example connects the POSITION event with method OnPosition, which extracts the new position from the notification arguments and stores it for the scroll bar. It also displays the new position and the event type for POSITION, which is to be 4:

```
::class MyDlgClass subclass UserDialog inherit MessageExtensions

::method InitDialog
  self~InitDialog:super(...)
  self~ConnectScrollBarNotify("MYSCROLL", "POSITION")

::method OnPosition
  use arg ev_pos, hnd
  pos = ev_pos % "10000"~x2d
  self~GetScrollBar("MYSCROLL")~SetPos(pos,1)   /* redraw scroll bar
                                                   with new position */
  say "Pos=" pos ", event code to verify=" BinaryAnd(ev_pos, ,
    "0x0000FFFF") "(expected 4)"
```

1. The method can only be called after the scroll bar was created by Windows. A good location for this connection is the InitDialog method.
2. The event-handling methods receive two arguments: an event-position pair and the handle to the scroll bar. You can retrieve the scroll bar position by extracting the high-order word.

   Example:

   ```
   ::method Handler
     use arg ev_pos, handle
     position = ev_pos % "10000"~x2d
   ```

   If the user changed the scroll bar position, you must set the scroll bar position with SetPos to keep the selected position.

13.12. ConnectTabNotify

```
>>-aMessageExtensions~ConnectTabNotify(--id--,--"--+-KEYDOWN-----+--"-->
                                                   +-SELCHANGE---+
                                                   +-SELCHANGING-+

>--+---------------+--)---------------------------------------------><
   +-,--msgToRaise-+
```

The ConnectTabNotify method connects a particular WM_NOTIFY message for a tab control with a method. The WM_NOTIFY message informs the dialog that an event has occurred with regard to the tab control.

Arguments:

The arguments are:

id

The ID of the tab control of which a notification is to be connected to a method.

event

The event to be connected with a method:

KEYDOWN

A key has been pressed while the tab control was focused.

SELCHANGE

Another tab has been selected in the tab control. This method is called after the selection was changed.

SELCHANGING

Another tab has been selected in the tab control. This method is called before the selection is changed.

msgToRaise

The message that is to be sent whenever the specified notification is received from the tab control. Provide a method with a matching name. If you omit this argument, the event is preceded by `On`.

Return value:

The return codes are:

0

No error detected.

-1

The resource ID could not be resolved or the event argument is incorrect.

1

The messages was not connected correctly.

Example:

The following example invokes method OnSelChange whenever another tab is selected in the tab control PAGE:

```
::class MyDlgClass subclass UserDialog inherit MessageExtensions

::method Init
  self~init:super(...)
  self~ConnectTabNotify("PAGE", "SELCHANGE")
```

Notes:

1. Connections are usually placed in the Init or InitDialog method. If both methods are defined, use init as the place for this connection - but not before init:super has been called.
2. The event-handling method that is connected to KEYDOWN receives two arguments: the control ID of the tab control and the virtual key code that has been pressed. Use the method KeyName of the VirtualKeyCodes class to get the name of the key. Note that your class must inherit from the VirtualKeyCodes class to use the KeyName method. Example:

   ```
   ::method OnKeyDown
     use arg id, vkey
     say "Key" self~KeyName(vkey) "was pressed."
   ```

3. All other event-handling methods receive two arguments: the ID of the tab control (extract the low-order word) and the handle to the tab control. Example:

   ```
   ::method Handler
     use arg ev_id, handle
   ```

```
id = BinaryAnd(ev_id, "0x0000FFFF")
```

13.13. ConnectSliderNotify

```
>>-aMessageExtensions~ConnectSliderNotify(--id--,--"--+-UP-------+--"-->
                                                      +-DOWN-----+
                                                      +-TOP------+
                                                      +-BOTTOM---+
                                                      +-PAGEUP---+
                                                      +-PAGEDOWN-+
                                                      +-DRAG-----+
                                                      +-POSITION-+
                                                      +-ENDTRACK-+

>--+---------------+--)---------------------------------------><
   +-,--msgToRaise-+
```

The ConnectSliderNotify method connects a particular WM_NOTIFY message for a slider control, which is also called a track bar, with a method. The WM_NOTIFY message informs the dialog that an event has occurred with regard to the slider control.

Arguments:

The arguments are:

id

The ID of the slider control of which a notification is to be connected to a method.

event

The event to be connected with a method:

UP

The Up or right key has been pressed.

DOWN

The Down or left key has been pressed.

TOP

The Home key has been pressed.

BOTTOM

The End key has been pressed.

PAGEUP

The PgUp key has been pressed.

PAGEDOWN

The PgDn key has been pressed.

DRAG

The slider has been moved.

POSITION

The left mouse button has been released, following a DRAG notification.

ENDTRACK

The slider movement is completed, that is, the appropriate key or mouse button has been released.

msgToRaise

The message that is to be sent whenever the specified notification is received from the slider control. Provide a method with a matching name. If you omit this argument, the event is preceded by `On`.

Return value:

The return codes are:

0

No error detected.

-1

The resource ID could not be resolved or the event argument is incorrect.

1

The messages was not connected correctly.

Example:

The following example connects the POSITION event (release mouse button after dragging) with method PosSet, which extracts the new slider position from the notification arguments and displays it together with the event type for POSITION, which is to be 4:

```
::class MyDlgClass subclass UserDialog inherit MessageExtensions

::method InitDialog
  self~InitDialog:super(...)
  self~ConnectSliderNotify("MYSLIDER", "POSITION", PosSet)

::method PosSet
  use arg ev_pos, hnd
```

```
pos = ev_pos % "10000"~x2d
say "Pos=" pos ", event code to verify=" BinaryAnd(ev_pos, ,
  "0x0000FFFF") "(expected 4)"
```

Notes:

1. The method can only be called after the slider was created by Windows. A good location for this connection is the InitDialog method.
2. The event-handling methods receive two arguments: an event-position pair and the handle to the slider control. For some events, you can retrieve the slider position by extracting the high-oder word. Example:

```
::method Handler
  use arg ev_pos, handle
  position = ev_pos % "10000"~x2d
```

Chapter 14. AdvancedControls Class

The AdvancedControls class provides methods to add and use the Win32 controls tree view control, list view control, tab control, slider control, and progress bar. It also provides methods to retrieve a specific object for any dialog control.

To use the methods defined by this mixin class, you must inherit from this class by specifying the INHERIT option for the ::CLASS directive in the class declaration. For example:

```
::class NewWin32Dialog SUBCLASS UserDialog INHERIT AdvancedControls
```

Requires:

The AdvancedControls class requires the class definition file `oodwin32.cls`:

```
::requires oodwin32.cls
```

Methods:

Instances of the AdvancedControls class implement the methods listed in the AdvancedControls Instance Methods table.

Table 14-1. AdvancedControls Instance Methods

Method...	...on page
AddListControl	AddListControl
AddProgressBar	AddProgressBar
AddSliderControl	AddSliderControl
AddTabControl	AddTabControl
AddTreeControl	AddTreeControl
ConnectListControl	ConnectListControl
ConnectSliderControl	ConnectSliderControl
ConnectTreeControl	ConnectTreeControl
GetButtonControl	GetButtonControl
GetCheckControl	GetCheckControl
GetComboBox	GetComboBox
GetEditControl	GetEditControl
GetListBox	GetListBox
GetListControl	GetListControl
GetProgressBar	GetProgressBar
GetRadioControl	GetRadioControl
GetScrollBar	GetScrollBar
GetSliderControl	GetSliderControl
GetStaticControl	GetStaticControl
GetTabControl	GetTabControl
GetTreeControl	GetTreeControl

14.1. GetStaticControl

```
>>-anAdvancedControl~GetStaticControl(--id--+-------------+--)-><
                                            +-,--category-+
```

The GetStaticControl method returns an object of the StaticControl class that is assigned to the static dialog item with the specified ID. The StaticControl class provides methods to query and manipulate static dialog items like static text, group boxes, or frames. The static controls must have a positive ID.

Arguments:

The arguments are:

id

The ID of the static dialog item.

category

The number of the category dialog page containing the requested dialog item. This argument must only be specified for category dialogs.

Return value:

An object of the StaticControl class or .Nil if the requested dialog item does not exist.

Example:

The following example requests an object of dialog item ITEM7 and, if the dialog item exists, resizes it, changes the displayed text, and sets another background and foreground color:

```
::class MyDlgClass subclass UserDialog inherit AdvancedControls

::method ReArrange
  di = self~GetStaticControl("ITEM7")
  if di == .Nil then return
  di~Resize(100, 25, "HIDE")
  di~Title="Processing layout update!"
  di~SetColor(7,4)
  di~Show
  ...
```

Note: GetStaticControl connects an Object Rexx object with a Windows object. If the object does not exist, the NIL object is returned. Therefore, this method can only be applied after the Windows dialog has been created (after the invocation of StartIt).

14.2. GetEditControl

```
>>-anAdvancedControl~GetEditControl(--id--+-------------+--)--->&lt;
                                          +-,--category-+
```

The GetEditControl method returns an object of the EditControl class that is assigned to the entry line with the specified ID. The EditControl class (see EditControl Class) provides methods to query and manipulate edit controls.

Arguments:

The arguments are:

id

The ID of the edit control.

category

The number of the category dialog page containing the requested edit control. This argument must only be specified for category dialogs.

Return value:

An object of the StaticControl class or .Nil if the requested edit control does not exist.

Example:

The following example gets an object of the EditControl class and checks whether the NAME entry line is empty. If the name field is empty, the dialog is not valid.

```
::class MyDlgClass subclass UserDialog inherit AdvancedControls

::method Validate
  di = self~GetEditControl("NAME")
  if di == .Nil then return 0
  if di~Title~space(0) \="" then return 1
```

Note: GetEditControl connects an Object Rexx object with a Windows object. If the object does not exist, the NIL object is returned. Therefore, this method can only be applied after the Windows dialog has been created (after the invocation of StartIt).

14.3. GetButtonControl

```
>>-anAdvancedControl~GetButtonControl(--id--+-------------+--)->&lt;
                                            +-,--category-+
```

The GetButtonControl method returns an object of the ButtonControl class that is assigned to the push button with the specified ID. The ButtonControl class (see ButtonControl Class) provides methods to query and manipulate push buttons.

Arguments:

The arguments are:

id

The ID of the push button.

category

The number of the category dialog page containing the requested push button. This argument must only be specified for category dialogs.

Return value:

An object of the ButtonControl class or .Nil if the requested push button does not exist.

Example:

The following example displays the current state of the OK button by retrieving an object of the Button class and calling the State method:

```
::class MyDlgClass subclass UserDialog inherit AdvancedControls

::method CurrentState
  di = self~GetButtonControl(1)
  if di == .Nil then return 0
  say "State is" di~State
```

Note: GetButtonControl connects an Object Rexx object with a Windows object. If the object does not exist, the NIL object is returned. Therefore, this method can only be applied after the Windows dialog has been created (after the invocation of StartIt).

14.4. GetRadioControl

```
>>-anAdvancedControl~GetRadioControl(--id--+-------------+--)--><
                                           +-,--category-+
```

The GetRadioControl method returns an object of the RadioButton class that is assigned to the radio button with the specified ID. The RadioButton class (see RadioButton Class) provides methods to query and manipulate radio buttons.

Arguments:

The arguments are:

id

The ID of the radio button.

category

The number of the category dialog page containing the requested radio button. This argument must only be specified for category dialogs.

Return value:

An object of the RadioButton class or .Nil if the requested radio button does not exist.

Example:

The following example displays a message when radio button CHOICE1 is selected:

```
::class MyDlgClass subclass UserDialog inherit AdvancedControls

::method CurrentState
  di = self~GetRadioControl("CHOICE1")
  if di == .Nil then return 0
  id di~IsChecked = "CHECKED" then say "The radio button is selected!"
```

Note: GetRadioControl connects an Object Rexx object with a Windows object. If the object does not exist, the NIL object is returned. Therefore, this method can only be applied after the Windows dialog has been created (after the invocation of StartIt).

14.5. GetCheckControl

```
>>-anAdvancedControl~GetCheckControl(--id--+-------------+--)-->&lt;
                                           +-,--category-+
```

The GetCheckControl method returns an object of the CheckBox class that is assigned to the check box with the specified ID. The CheckBox Class (see CheckBox Class) provides methods to query and manipulate check boxes.

Arguments:

The arguments are:

id

The ID of the check box.

category

The number of the category dialog page containing the requested check box. This argument must only be specified for category dialogs.

Return value:

An object of the CheckBox class or .Nil if the requested check box does not exist.

Example:

The following example displays a message when check box CHOICE1 is checked:

```
::class MyDlgClass subclass UserDialog inherit AdvancedControls

::method CurrentState
  di = self~GetCheckControl("CHOICE1")
  if di == .Nil then return 0
  if di~IsChecked = "CHECKED" then say "The check box is checked!"
```

Note: GetCheckControl connects an Object Rexx object with a Windows object. If the object does not exist, the NIL object is returned. Therefore, this method can only be applied after the Windows dialog has been created (after the invocation of StartIt).

14.6. GetListBox

```
>>-anAdvancedControl~GetListBox(--id--+-------------+--)------><
                                      +-,--category-+
```

The GetListBox method returns an object of the ListBox class that is assigned to the list box with the specified ID. The ListBox class (see ListBox Class) provides methods to query and manipulate list boxes.

Arguments:

The arguments are:

id

The ID of the list box.

category

The number of the category dialog page containing the requested list box. This argument must only be specified for category dialogs.

Return value:

An object of the ListBox class or .Nil if the requested list box does not exist.

Example:

The following example removes all entries of list box AREAS and adds several new entries. Entry City will be preselected. Object "di" is connected to list box AREAS.

```
::class MyDlgClass subclass UserDialog inherit AdvancedControls

::method UpdateList
  di = self~GetListBox("AREAS")
  if di == .Nil then return 0
  di~DeleteAll
  di~Add("Town")
  di~Add("City")
  di~Add("Green")
  di~Add("Forest")
  di~Select("City")
```

Note: GetListBox connects an Object Rexx object with a Windows object. If the object does not exist, the NIL object is returned. Therefore, this method can only be applied after the Windows dialog has been created (after the invocation of StartIt).

14.7. GetComboBox

```
>>-anAdvancedControl~GetComboBox(--id--+-------------+--)------><
                                       +-,--category-+
```

The GetComboBox method returns an object of the ComboBox class that is assigned to the list box with the specified ID. The ComboBox class (see ComboBox Class) provides methods to query and manipulate combo boxes.

Arguments:

The arguments are:

id

The ID of the combo box.

category

The number of the category dialog page containing the requested combo box. This argument must only be specified for category dialogs.

Return value:

An object of the ComboBox class or .Nil if the requested combo box does not exist.

Example:

The following example removes all entries of combo box AREAS and adds several new entries. Entry "City" will be preselected. Object "di" is connected to combo box AREAS.

```
::class MyDlgClass subclass UserDialog inherit AdvancedControls

::method UpdateList
  di = self~GetComboBox("AREAS")
  if di == .Nil then return 0
  di~DeleteAll
  di~Add("Town")
  di~Add("City")
  di~Add("Green")
  di~Add("Forest")
  di~Select("City")
```

Note: GetComboBox connects an Object Rexx object with a Windows object. If the object does not exist, the NIL object is returned. Therefore, this method can only be applied after the Windows dialog has been created (after the invocation of StartIt).

14.8. GetScrollBar

```
>>-anAdvancedControl~GetScrollBar(--id--+-------------+--)----->&lt;
                                        +-,--category-+
```

The GetScrollBar method returns an object of the ScrollBar class that is assigned to the scroll bar with the specified ID. The ScrollBar class (see ScrollBar Class) provides methods to query and manipulate scroll bars.

Arguments:

The arguments are:

id

The ID of the scroll bar.

category

The number of the category dialog page containing the requested scroll bar. This argument must only be specified for category dialogs.

Return value:

An object of the ScrollBar class or .Nil if the requested scroll bar does not exist.

Example:

The following example sets a new range and a new position for scroll bar HORSB. Object "di" is connected to scroll bar HORSB.

```
::class MyDlgClass subclass UserDialog inherit AdvancedControls

::method FocusPage
  di = self~GetScrollBar("HORSB")
  if di == .Nil then return 0
  di~SetRange(0, 1000, 0)
  di~SetPos(500, 1)
```

Note: GetScrollBar connects an Object Rexx object with a Windows object. If the object does not exist, the NIL object is returned. Therefore, this method can only be applied after the Windows dialog has been created (after the invocation of StartIt).

14.9. GetTreeControl

```
>>-anAdvancedControl~GetTreeControl(--id--+-------------+--)--->
                                          +-,--category-+
```

The GetTreeControl method returns an object of the TreeControl class that is assigned to the tree view with the specified ID. The TreeControl class (see TreeControl Class) provides methods to query and manipulate tree views.

Arguments:

The arguments are:

id

The ID of the tree view.

category

The number of the category dialog page containing the requested tree view. This argument must only be specified for category dialogs.

Return value:

An object of the TreeControl class or .Nil if the requested tree view does not exist.

Example:

The following example initializes tree view 101 by sending message ADD to object "tc", which is assigned to 101 by using GetTreeControl:

```
::class MyDlgClass subclass UserDialog inherit AdvancedControls
```

```
::method InitDialog
  tc = self~GetTreeControl(101)
  if tc == .Nil then return
  tc~Add("Root 1")
  tc~Add(  ,"Item 1")
  tc~Add(  ,"Item 2")
  tc~Add(  ,"Item 3")
  tc~Add("Root 2", , ,"EXPANDED")
  tc~Add(  ,"Item 4", , ,"BOLD")
  tc~Add(  ,"Item 5")
  tc~Add(  ,"Subroot")
  tc~Add(  ,    ,"Item 6",3)
```

Note: GetTreeControl connects an Object Rexx object with a Windows object. If the object does not exist, the NIL object is returned. Therefore, this method can only be applied after the Windows dialog has been created (after the invocation of StartIt).

14.10. GetListControl

```
>>-anAdvancedControl~GetListControl(--id--+-------------+--)---><
                                          +-,--category-+
```

The GetListControl method returns an object of the ListControl class that is assigned to the list view with the specified ID. The ListControl class (see ListControl Class) provides methods to query and manipulate list views.

Arguments:

The arguments are:

id

The ID of the list view.

category

The number of the category dialog page containing the requested list view. This argument must only be specified for category dialogs.

Return value:

An object of the ListControl class or .Nil if the requested list view does not exist.

Example:

The following example initializes list view 101 by sending message ADD to object lc, which is assigned to 101 by using GetListControl:

```
::class MyDlgClass subclass UserDialog inherit AdvancedControls
```

```
::method InitDialog
  lc = self~GetListControl(101)
  if lc == .Nil then return
  lc~~Add(101222)~~Add(,"Smith")~~Add(, ,"John")
  lc~~Add(101223)~~Add(,"Michael")~~Add(, ,"Carl")
```

Note: GetListControl connects an Object Rexx object with a Windows object. If the object does not exist, the NIL object is returned. Therefore, this method can only be applied after the Windows dialog has been created (after the invocation of StartIt).

14.11. GetProgressBar

```
>>-anAdvancedControl~GetProgressBar(--id--+-------------+--)--->&lt;
                                          +-,--category-+
```

The GetProgressBar method returns an object of the ProgressBar Control class that is assigned to the progress bar with the specified ID. The ProgressBar class (see ProgressBar Class) provides methods to query and manipulate progress bars.

Arguments:

The arguments are:

id

The ID of the progress bar.

category

The number of the category dialog page containing the requested progress bar. This argument must only be specified for category dialogs.

Return value:

An object of the ProgressBar class or .Nil if the requested progress bar does not exist.

Example:

The following example initializes and modifies progress bar PROGRESS by sending messages to the object that is returned by GetProgressBar:

```
::class MyDlgClass subclass UserDialog inherit AdvancedControls

::method InitDialog
  pb = self~GetProgressBar("PROGRESS")
  if pb == .Nil then return
  pb~setstep(50)
  pb~setrange(,500)
```

```
::method UpdateProgress
  use arg amount
  self~GetProgressBar("PROGRESS")~SetPos(amount)
```

Note: GetProgressBar connects an Object Rexx object with a Windows object. If the object does not exist, the NIL object is returned. Therefore, this method can only be applied after the Windows dialog has been created (after the invocation of StartIt).

14.12. GetSliderControl

```
>>-anAdvancedControl~GetSliderControl(--id--+-------------+--)-><
                                            +-,--category-+
```

The GetSliderControl method returns an object of the SliderControl class that is assigned to the track bar with the specified ID. The SliderControl class (see SliderControl Class) provides methods to query and manipulate track bars.

Arguments:

The arguments are:

id

The ID of the track bar.

category

The number of the category dialog page containing the requested track bar. This argument must only be specified for category dialogs.

Return value:

An object of the SliderControl class or .Nil if the requested track bar does not exist.

Example:

The following example initializes track bar 103:

```
::class MyDlgClass subclass UserDialog inherit AdvancedControls

::method InitTheSlider
  sl = self~GetSliderControl(103)
  if sl == .Nil then return
  no = 0; yes = 1
  sl~ClearSelRange(no)
  sl~SetMax(200,no)
  sl~SetTickFrequency(50)
  sl~SetTickAt(75)
```

```
sl~SetSelStart(20, no)
sl~SetSelEnd(180, yes)
sl~Pos = 167
```

Note: GetSliderControl connects an Object Rexx object with a Windows object. If the object does not exist, the NIL object is returned. Therefore, this method can only be applied after the Windows dialog has been created (after the invocation of StartIt).

14.13. GetTabControl

```
>>-anAdvancedControl~GetTabControl(--id--+-------------+--)----><
                                         +-,--category-+
```

The GetTabControl method returns an object of the TabControl class that is assigned to the tab control with the specified ID. The TabControl class (see TabControl Class) provides methods to query and manipulate tab controls.

Arguments:

The arguments are:

id

The ID of the tab control.

category

The number of the category dialog page containing the requested tab control. This argument must only be specified for category dialogs.

Return value:

An object of the TabControl class or .Nil if the requested tab control does not exist.

Example:

The following example initializes tab control PAGES to have five tabs:

```
::class MyDlgClass subclass UserDialog inherit AdvancedControls

::method InitDialog
  self~GetTabControl("PAGES")~AddSequence("Design","Implementation", ,
  "Test","Review","Release")
```

Note: GetTabControl connects an Object Rexx object with a Windows object. If the object does not exist, the NIL object is returned. Therefore, this method can only be applied after the Windows dialog has been created (after the invocation of StartIt).

14.14. ConnectTreeControl

```
>>-anAdvancedControl~ConnectTreeControl(--id--+------------------+--)-><
                                              +-,--attributeName-+
```

The ConnectTreeControl method creates a new attribute and connects it to the tree view *id*. The attribute has to be synchronized manually with the tree view. You can do this globally using the SetData and GetData methods or methods provided by the TreeControl class. A tree view can contain many items. When the dialog data is set, the first tree view item containing the same text as the text stored in the connected attribute, is selected. When the data is received, the attribute receives the text of the selected tree view item. Usually, the connection is made automatically and you do not have to use this method.

Arguments:

The arguments are:

id

The ID of the tree view that you want to connect.

attributeName

An unused valid Rexx symbol because an attribute with exactly this name is added to the dialog object by this method. Blank spaces, ampersands (&), and colons (:) are removed from the attributeName.

If this argument is omitted, is not valid, or already exists, the following occurs:

- If the ID is numeric, an attribute with the name DATAid is used, where *id* is the value of the first argument.
- If the ID is symbolic, the attribute is named as the ID.

Example:

In the following example, the tree view with ID 202 is associated with the attribute FileName. Then TEST.REX is assigned to the newly created attribute. Then the dialog is executed, which preselects TEST.REX in the tree view, if it exits. After the dialog is terminated, the selected entry of the tree view is copied to the attribute FileName.

```
MyDialog~ConnectTreeControl(202, "FileName")
MyDialog~FileName="TEST.REX"
MyDialog~Execute("SHOWTOP")
say MyDialog~FileName
```

14.15. ConnectListControl

```
>>-anAdvancedControl~ConnectListControl(--id--+------------------+--)-><
```

```
                                          +-,--attributeName-+
```

The ConnectListControl method creates a new attribute and connects it to the list view *id*. The *attributeName* is a string containing the numbers of the selected lines. The numbers are separated by blanks. Therefore, if value of the attribute after GetData is "3 5 6", the third, fifth, and sixth items are currently selected, or will be selected when SetData is executed. For further information, refer to ConnectTreeControl.

Example:

In the following example, the list view with ID 202 is associated with the attribute Customers. The first, 14th, and 29th entries in the list are preselected.

```
MyDialog~ConnectListControl(202, "Customers")
MyDialog~Customers="1 14 29"
```

14.16. ConnectSliderControl

```
>>-anAdvancedControl~ConnectSliderControl(--id--+------------------+--)-><
                                                +-,--attributeName-+
```

The ConnectSliderControl method creates a new attribute and connects it to the track bar *id*. The *attributeName* is the numerical position of the slider. For further information, refer to ConnectTreeControl.

14.17. ConnectTabControl

```
>>-anAdvancedControl~ConnectTabControl(--id--+------------------+--)-><
                                             +-,--attributeName-+
```

The ConnectTabControl method creates a new attribute and connects it to the tab control *id*. The *attributeName* is the text of the active tab. For further information, refer to ConnectTreeControl.

14.18. AddTreeControl

```
>>-anAdvancedControl~AddTreeControl(--id--,--+---------------+--,--x--,--y-->
                                             +-attributeName-+

>--,--cx--,--cy--+-------------------------------+--)--------------------><
                 |       +-------------------+    |
                 |       V                   |    |
                 +-,--"----+-ATROOT--------+-+--"-+
                           +-BUTTONS-------+
```

```
+-LINES---------+
+-EDIT----------+
+-HSCROLL-------+
+-VSCROLL-------+
+-SHOWSELALWAYS-+
+-ALL-----------+
+-NODRAG--------+
+-GROUP---------+
+-HIDDEN--------+
+-NOTAB---------+
+-NOBORDER------+
```

The AddTreeControl method adds a tree view to the dialog and connects it with a data attribute. For further information on tree view controls, refer to TreeControl Class.

Arguments:

The arguments are:

id

A unique identifier assigned to the control. You need the ID to refer to this control in other methods.

attributeName

The name of the data attribute associated with the dialog item. See attributeName to get information on what happens when this argument is omitted.

x, y

The position of the upper left corner of the control relative to the dialog, in dialog units.

cx, xy

The width and height of the dialog item, in dialog units.

options

This argument determines the behavior and style of the dialog item and can be one or more of the following, separated by blanks:

ATROOT

The tree view has lines linking child items to the root of the hierarchy.

BUTTONS

The tree view adds a button to the left of each parent item.

LINES

The tree view has lines linking child items to their corresponding parent items.

EDIT

The tree view allows the user to edit the labels of tree view items. To store the edited text, you must connect a method to the ENDEDIT notification or connect the DEFAULTEDIT event handler (see ConnectTreeNotify).

HSCROLL

The tree view supports a horizontal scroll bar.

VSCROLL

The tree view supports a vertical scroll bar.

SHOWSELALWAYS

A selected item remains selected when the tree view loses focus.

ALL

The options ATROOT, BUTTONS, LINES, EDIT, HSCROLL, and SHOWSELALWAYS are all applied.

NODRAG

The tree view is prevented from sending "begin drag" notifications.

GROUP

The first control of a group of controls in which the user can move from one control to the next with the arrow key.

HIDDEN

The control is initially hidden.

NOTAB

The tab key cannot be used to move to this control.

NOBORDER

No border is drawn around the control.

Example:

The following example creates a tree view at position x=100 and y=80 and with a size of width=40 and height=120. The ID of the tree is 555 and its data is associated with attribute BRANCH. The tree view uses lines for child items and roots, displays a button to the left of each parent item, and supports item editing.

```
MyDialog~AddTreeControl(555, "Branch", 100, 80, 40, 120, ,
"LINES BUTTON EDIT ATROOT")
```

14.19. AddListControl

```
>>-anAdvancedControl~AddListControl(--id--,--+---------------+--,--x,--y-->
                                             +-attributeName-+

>--,--cx--,--cy--+-------------------------------+--)------------------><
                 |       +-------------------+    |
                 |       V                   |    |
                 +-,--"----+-ICON----------+-+--"-+
                           +-SMALLICON-----+
                           +-LIST----------+
                           +-REPORT--------+
                           +-ALIGNLEFT-----+
                           +-ALIGNTOP------+
                           +-AUTOARRANGE---+
                           +-ASCENDING-----+
                           +-DESCENDING----+
                           +-EDIT----------+
                           +-HSCROLL-------+
                           +-VSCROLL-------+
                           +-NOSCROLL------+
                           +-NOHEADER------+
                           +-NOSORTHEADER--+
                           +-NOWRAP--------+
                           +-SINGLESEL-----+
                           +-SHOWSELALWAYS-+
                           +-SHAREIMAGES---+
                           +-GROUP---------+
                           +-HIDDEN--------+
                           +-NOTAB---------+
                           +-NOBORDER------+
```

The AddListControl method adds a list view to the dialog and connects it with a data attribute. For further information on list view controls, refer to ListControl Class.

Arguments:

The arguments are:

id

A unique identifier assigned to the control. You need the ID to refer to this control in other methods.

attributeName

The name of the data attribute associated with the dialog item. See ConnectTreeControl to get information on what happens when this argument is omitted.

x, y

The position of the upper left corner of the control relative to the dialog, in dialog units.

cx, xy

The width and height of the dialog item, in dialog units.

options

This argument determines the behavior and style of the dialog item and can be one or more of the following, separated by blanks:

ICON

Use the icon view. Each item appears as a full-sized icon with a label below it. The user can drag the items to any location in the list view control.

SMALLICON

Use the small-icon view. Each item appears as a small icon with a label to the right of it. The user can drag the items to any location in the list view control.

LIST

Use the list view. Each item appears as a small icon with a label to the right of it. Items are arranged in columns and cannot be moved by the user.

REPORT

Use the report view. Each item appears on a separate line with information arranged in columns. The leftmost column contains the small icon and label, and subsequent columns contain subitems.

ALIGNLEFT

In icon and small-icon views, the items are left-aligned.

ALIGNTOP

In icon and small-icon views, the items are aligned with the top of the control.

AUTOARRANGE

In icon and small-icon views, the icons are always automatically arranged.

ASCENDING

Sorts items by item text in ascending order.

DESCENDING

Sorts items by item text in descending order.

EDIT

The list view allows the user to edit the list view items. To store the edited text, you must connect a method to the ENDEDIT notification or you connect the DEFAULTEDIT event handler (see ConnectTreeNotify).

HSCROLL

The list view supports a horizontal scroll bar.

VSCROLL

The list view supports a vertical scroll bar.

NOSCROLL

Disables scrolling.

NOHEADER

No column header is displayed in the report view. By default, columns have headers in the report view.

NOSORTHEADER

Specifies that column headers do not work like buttons. This option is useful if clicking a header in the report view does not carry out an action.

NOWRAP

Displays item text on a single line in the icon view. By default, the item text can wrap in the icon view.

SINGLESEL

Allows only one item to be selected at a time. By default, several items can be selected.

SHOWSELALWAYS

Specifies that a selected item remains selected when the list view loses focus.

DEFAULTEDIT

Connects the notification that label editing has been started and ended with a predefined event-handling method. This method extracts the newly entered text from the notification and modifies the item of which the label was edited. If this event is not connected you must provide your own event-handling method and connect it with the BEGINEDIT and ENDEDIT events. Otherwise, the edited text is lost and the item remains unchanged.

SHAREIMAGES

The control does not take ownership of the image lists assigned to it. This option enables an image list to be used with several list controls.

GROUP

Specifies the first control of a group of control in which the user can move from one control to the next with the arrow keys.

HIDDEN

The control is initially hidden.

NOTAB

The tab key cannot be used to move to this control.

NOBORDER

No border is drawn around the control.

Example:

The following example creates a list view at position x=100 and y=80 and with a size of width=40 and height=120. The list view with ID 555 is a report view with items sorted in ascending order. It supports item editing and column headers do not behave like buttons. Its data is associated with attribute EMPLOYEES.

```
MyDialog~AddListControl(555, "EMPLOYEES", 100, 80, 40, 120, ,
"REPORT ASCENDING EDIT NOSORTHEADER")
```

14.20. AddProgressBar

```
>>-anAdvancedControl~AddProgressBar(--id--,--x--,--y--,--cx--,--cy-->

>--+---------------------------+--)-------------------------------><
   |       +--------------+    |
   |       V              |    |
   +-,--"----+-VERTICAL-+-+--"-+
             +-SMOOTH---+
             +-BORDER---+
```

The AddProgressBar method adds a progress bar to the dialog and connects it with a data attribute. For further information on progress bar controls, refer to ProgressBar Class.

Arguments:

The arguments are:

id

A unique identifier assigned to the control. You need the ID to refer to this control in other methods.

x, y

The position of the upper left corner of the control relative to the dialog, in dialog units.

cx, xy

The width and height of the dialog item, in dialog units.

options

This argument determines the behavior and style of the dialog item and can be one or more of the following:

VERTICAL

The progress bar is oriented vertically.

SMOOTH

The progress bar is incremented smoothly.

BORDER

A border is drawn around the progress bar.

If you omit this argument, the progress bar is oriented horizontally.

Example:

The following example creates a progress bar with ID DONE at the bottom of the dialog. The progress bar is as wide as the dialog.

```
MyDialog~AddProgressBar("DONE",10,MyDialog~sizey-16,MyDialog~sizex-20,12)
```

14.21. AddSliderControl

```
>>-anAdvancedControl~AddSliderControl(--id--,--+---------------+--,--x-->
                                               +-attributeName-+

>--,--y--,--cx--,--cy--+--------------------------------+--)---------><
                       |       +--------------------+    |
                       |       V                    |    |
                       +-,--"----+-AUTOTICKS------+-+--"-+
                                 +-NOTICKS--------+
                                 +-HORIZONTAL-----+
                                 +-VERTICAL-------+
                                 +-TOP------------+
                                 +-BOTTOM---------+
                                 +-LEFT-----------+
                                 +-RIGHT----------+
                                 +-BOTH-----------+
                                 +-ENABLESELRANGE-+
                                 +-GROUP----------+
                                 +-HIDDEN---------+
                                 +-NOTAB----------+
                                 +-NOBORDER-------+
```

The AddSliderControl method adds a slider control (track bar) to the dialog and connects it with a data attribute. For further information on slider controls, refer to SliderControl Class.

Arguments:

The arguments are:

id

A unique identifier assigned to the control. You need the ID to refer to this control in other methods.

attributeName

The name of the data attribute associated with the dialog item. See attributeName to get information on what happens when this argument is omitted.

x, y

The position of the upper left corner of the control relative to the dialog, in dialog units.

cx, xy

The width and height of the dialog item, in dialog units.

options

This argument determines the behavior and style of the dialog item and can be one or more of the following:

AUTOTICKS

Creates a slider that has a tick mark for each increment in its range of values. These tick marks are automatically added when an application calls the InitRange method. You cannot use the SetTickAt and SetTickFrequency methods to specify the position of the tick marks when you use this option.

NOTICKS

Creates a slider that does not display tick marks.

HORIZONTAL

Orients the slider horizontally. This is the default orientation.

VERTICAL

Orients the slider vertically.

TOP

Displays tick marks along the top of a horizontal slider.

BOTTOM

Displays tick marks along the bottom of a horizontal slider. This option can be used together with the TOP option to display tick marks on both sides of the slider control.

LEFT

Displays tick marks along the left of a vertical slider.

RIGHT

Displays tick marks along the right of a vertical slider. This option can be used together with the LEFT option to display tick marks on both sides of the slider control.

BOTH

Displays tick marks on both sides of the slider in any direction.

ENABLESELRANGE

Displays a selection range. If you set this option, the tick marks at the starting and ending positions of a selection range are displayed as triangles and the selection range is highlighted. This can be used, for example, to indicate a preferred selection.

GROUP

Specifies the first control of a group of control in which the user can move from one control to the next with the arrow keys.

HIDDEN

The control is initially hidden.

NOTAB

The tab key cannot be used to move to this control.

NOBORDER

No border is drawn around the control.

Example:

The following example creates a vertical slider control at the right border of the dialog. The slider with ID PRESSURE displays automatic tick marks on both sides. Its position is associated with attribute PRESSURE.

```
MyDialog~AddSliderControl("PRESSURE", , MyDialog~sizeX-30, 15, 20, ,
MyDialog~sizeY-30, "VERTICAL BOTH AUTOTICKS")
```

14.22. AddTabControl

```
>>-anAdvancedControl~AddTabControl(--id--,--+---------------+--,--x-->
                                            +-attributeName-+

>--,--y--,--cx--,--cy--+-----------------------------+--)---------><
                       |       +-----------------+    |
                       |       V                 |    |
                       +-,--"----+-ALIGNRIGHT--+-+--"-+
                                 +-BUTTONS-----+
                                 +-FIXED-------+
                                 +-FOCUSNEVER--+
                                 +-FOCUSONDOWN-+
```

```
+-ICONLEFT----+
+-LABELLEFT---+
+-MULTILINE---+
+-GROUP-------+
+-HIDDEN------+
+-NOTAB-------+
+-NOBORDER----+
```

The AddTabControl method adds a tab control to the dialog and connects it with a data attribute. For further information on tab controls, refer to TabControl Class.

Arguments:

The arguments are:

id

A unique identifier assigned to the control. You need the ID to refer to this control in other methods.

attributeName

The name of the data attribute associated with the dialog item. See ConnectTreeControl to get information on what happens when this argument is omitted.

x, y

The position of the upper left corner of the control relative to the dialog, in dialog units.

cx, xy

The width and height of the dialog item, in dialog units.

options

This argument determines the behavior and style of the dialog item and can be one or more of the following:

ALIGNRIGHT

Right-justifies tabs. By default, tabs are left-justified within a row.

BUTTONS

Modifies the appearance of the tabs to look like buttons.

FIXED

Makes all tabs equal in width. You cannot use this option with the ALIGNRIGHT option.

FOCUSNEVER

A tab never receives the input focus.

FOCUSONDOWN

A tab receives the input focus when clicked (typically with option BUTTONS).

ICONLEFT

Forces the icon to the left, but leaves the tab label centered. By default, the control centers the icon and label, with the icon being to the left of the label.

LABELLEFT

Left-aligns both the icon and the label.

MULTILINE

Causes a tab control to display several rows of tabs, enabling all tabs to be displayed at the same time.

GROUP

Specifies the first control of a group of control in which the user can move from one control to the next with the arrow keys.

HIDDEN

The control is initially hidden.

NOTAB

The tab key cannot be used to move to this control.

NOBORDER

No border is drawn around the control.

Example:

The following example creates a tab control with ID PAGES and multiline capability. Its data (the selected tab) is associated with attribute CURRENTPAGE.

```
MyDialog~AddTabControl("PAGES","CURRENTPAGE", 10, 120, 200, 20, ,
"MULTILINE FIXED")
```

Chapter 15. StaticControl Class

The StaticControl class provides methods to query and modify static controls, such as static text, group boxes, and frames. It inherits all methods of the DialogControl class (see DialogControl Class).

The StaticControl class requires the class definition file `oodwin32.cls`:

```
::requires "oodwin32.cls"
```

Use the GetStaticControl method (see GetStaticControl) to retrieve an object of the StaticControl class. To use this method, the static control must have a positive ID.

Chapter 16. EditControl Class

The EditControl class provides methods to query and modify edit controls, which are also called entry lines in the ooDialog documentation. It inherits all methods of the DialogControl class (see page DialogControl Class).

Use the GetEditControl method (see page GetEditControl) to retrieve an object of the EditControl class.

Requires:

The EditControl class requires the class definition file `oodwin32.cls`:

```
::requires "oodwin32.cls"
```

Methods:

Instances of the EditControl class implement the methods listed in the following EditControl Instance Methods table.

Table 16-1. EditControl Instance Methods

Method...	...on page
AddStyle	AddStyle
EnsureCaretVisibility	EnsureCaretVisibility
FirstVisibleLine	FirstVisibleLine
GetLine	GetLine
GetStyle	GetStyle
GetText	GetText
HideBalloon	HideBalloon
IsModified	IsModified
LineFromIndex	LineFromIndex
LineIndex	LineIndex
LineLength	LineLength
LineScroll	LineScroll
Lines	Lines
Margins	Margins
PasswordChar	PasswordChar
PasswordChar=	PasswordChar=
RemoveStyle	RemoveStyle
ReplaceStyle	ReplaceStyle
ReplaceSelText	ReplaceSelText
ScrollCommand	ScrollCommand
Select	Select
Selected	Selected
SetCue	SetCue

Method...	...on page
SetLimit	SetLimit
SetMargins	SetMargins
SetModified	SetModified
SetReadOnly	SetReadOnly
SetText	SetText
ShowBalloon	ShowBalloon

16.1. Selected

```
>>-anEditControl~Selected-------------------------------------><
```

The Selected method retrieves the indexes of the starting and ending character of the text selected. If the starting index equals the ending index, no text is selected and the index specifies the current cursor position. If the ending index is 0 and the starting index is 1, the entire text is selected.

Return value:

The one-based starting and ending index of the current selection, separated by a blank.

Example:

The following example displays the starting and ending index of the text selection of the edit control NAME. It then selects the entire text of the edit control.

```
edit = MyDialog~GetEditControl("NAME")
if edit == .Nil then return
parse value edit~Selected with start end
say "Starting index of selection is" start
say "Ending index of selection is" end
edit~Select(1,0)
```

16.2. Select

```
>>-anEditControl~Select(--start--,--end--)--------------------><
```

The Select method selects the text or sets the cursor position for the associated edit control. If the starting index equals the ending index, no text is selected and the cursor is set to the character at the specified index. If the ending index is 0 and the starting index is 1, the entire text is selected.

Arguments:

The arguments are:

start

A one-based index where the selection begins or the cursor is to be set.

end

A one-based index where the selection ends.

Example:

See Selected.

16.3. ScrollCommand

```
                                    +-UP--------+
>>-anEditControl~ScrollCommand(--"--+-----------+--"--+----------------+--)-><
                                    +-LEFT------+     +-,--repetitions-+
                                    +-DOWN------+
                                    +-RIGHT-----+
                                    +-PAGEUP----+
                                    +-PAGELEFT--+
                                    +-PAGEDOWN--+
                                    +-PAGERIGHT-+
```

The ScrollCommand method scrolls the associated edit control in a given direction.

Arguments:

The arguments are:

command

Specifies the direction and the step size of the scroll command. Possible values are:

UP or LEFT

Scrolls up one line. UP is the default.

DOWN or RIGHT

Scrolls down one line.

PAGEUP or PAGELEFT

Scrolls up one page.

PAGEDOWN or PAGERIGHT

Scrolls down one page.

repetitions

The number of times the scroll command is to be issued. If this argument is omitted, the scroll command is issued once.

Example:

The following example scrolls down 3 lines in the edit control:

```
edit = MyDialog~GetEditControl("NAME")
if edit == .Nil then return
edit~ScrollCommand("DOWN",3)
```

16.4. LineScroll

```
>>-anEditControl~LineScroll(--sChars--,--sLines--)------------><
```

The LineScroll method scrolls the text in a multiline edit control vertically or horizontally by the specified number of characters and lines.

Arguments:

The arguments are:

sChars

The number of characters to be scrolled horizontally.

sLines

The number of lines to be scrolled vertically.

Return value:

0 if the message is sent to a multiline edit control, or a non-zero value if the message is sent to a single-line edit control.

Example:

The following example scrolls an edit control by 50 characters and 35 lines:

```
edit~Scroll(50, 35)
```

Note: The edit control does not scroll vertically past the last line of text. If the current line, plus the number of lines specified by *sLines*, exceeds the total number of lines in the edit control, the last line of the edit control is scrolled to the top. The LineScroll message can, however, be used to scroll horizontally past the last character of a line.

16.5. EnsureCaretVisibility

```
>>-anEditControl~EnsureCaretVisibility------------------------><
```

The EnsureCaretVisibility method scrolls the edit control until the caret (cursor) is visible.

Return value:

0 if the object is associated with an existing edit control.

16.6. FirstVisibleLine

```
>>-anEditControl~FirstVisibleLine------------------------------><
```

The FirstVisibleLine method retrieves the one-based line number of the first line visible in a multiline edit control.

Return value:

The number of the first visible line, starting with 1.

Example:

For an example, refer to PasswordChar=.

16.7. IsModified

```
>>-anEditControl~IsModified-----------------------------------><
```

The IsModified method retrieves information on whether the edit control has been modified.

Return value:

1

The text in the edit control has been altered.

0

For all other cases.

Example:

(See also the SetText example.)

```
if edit~IsModified = 1 then MyDialog~Save
```

16.8. SetModified

```
>>-anEditControl~SetModified(--bool--)------------------------><
```

The SetModified method sets the flag to indicate whether the edit control has been modified.

Arguments:

The only argument is:

bool

1

The flag indicates that the text has been altered.

0

For all other cases.

Example:

In the following example, the Save method stores the dialog contents in a file and clears the modified flag. See also the SetText example.

```
::method Save
    /* write contents to file */
    edit = MyDialog~GetEditControl("TEXT")
    ...
    edit~SetModified(0)
```

16.9. Lines

```
>>-anEditControl~Lines----------------------------------------><
```

The Lines method retrieves the number of text lines of a multiline edit control.

Return value:

The number of text lines.

Example:

For an example, refer to LineIndex.

16.10. LineIndex

```
>>-anEditControl~LineIndex(--line--)--------------------------><
```

The LineIndex method retrieves the one-based character index of the beginning of the *line* in the associated edit control.

Arguments:

The only argument is:

line

The number of the line of which the starting index is to be retrieved. Line numbers are incremented by 1, starting with 1.

Return value:

The character index of the specified line. The first line starts at index 1.

Example:

The following example sets the text for edit control TEXT and displays the number of text lines (3), the starting index of the second line (carriage return, 0x0d, and line feed, 0x0a, which mark a line break, are also considered to be characters), and the length of the third line:

```
editControl = MyDialog~GetEditControl("TEXT")
if .nil == editControl then return

text = "It is easy to learn and easy to use." || '0d0a'x || -
       "Have fun with it!"                    || '0d0a'x || -
       "But don't over do it."

edit~setText(text)

say "Number of lines:" edit~lines
say "Line 2 begins at index" edit~LineIndex(2)
say "Length of 3rd line:" edit~LineLength(3)

Result: Number of lines: 3
        Line 2 begins at index 39
        Length of 3rd line: 21
```

16.11. LineFromIndex

```
>>-anEditControl~LineFromIndex(--index--)---------------------><
```

The LineFromIndex method retrieves the one-based line number that contains the character index *index*.

Arguments:

The only argument is:

index

The one-based character index contained in the line whose number is to be retrieved.

Return value:

The line number containing the specified character index. The first line starts at index 1. If the specified index exceeds the number of characters contained in the edit control or an invalid character index was specified, 0 is returned.

Example:

The following example displays the line in which character 55 is contained:

```
edit = MyDialog~GetEditControl("TEXT")
if edit == .Nil then return

"It is easy to learn and easy to use." || 13~d2c || 10~d2c ||,
"Have fun with it!"
say "Character 55 is contained in line" edit~LineFromIndex(55)

Result: Character 55 is contained in line 2
```

16.12. LineLength

```
>>-anEditControl~LineLength(--line--)-------------------------><
```

The LineLength method retrieves the number of characters contained in the *line* in the associated edit control.

Arguments:

The only argument is:

line

The number of the line of which the number of characters is to be retrieved. Line numbers are incremented by 1, starting with 1.

Return value:

The length of the given line.

Example:

For an example, refer to LineIndex.

16.13. GetLine

```
>>-anEditControl~GetLine--(--line--+--------------+--)--------->&lt;
                                   +-,--maxLength-+
```

The GetLine method retrieves the text string contained in the specified line.

Arguments:

The arguments are:

line

The one-based line number to be retrieved.

maxLength

The maximum number of characters to be retrieved. Object Rexx allocates the appropriate amount of memory to store the text string. If the line consists of more characters than fit in the memory, the text string is truncated. If you omit this argument, the maximum number of characters retrieved is 255.

Return value:

A text string or an empty string.

Example:

The following example stores all lines contained in edit control EDITOR in a stem. If a line consists of more than 1024 characters, it is truncated.

```
edit = MyDialog~GetEditControl("EDITOR")
if edit == .Nil then return
do i = 1 to edit~Lines
   lines.i = edit~GetLine(i, 1024)
end
```

Note: The carriage return and line-feed characters are not included in the returned text string. To get the contents of a single line edit control, use the GetText) method, or use `edit~GetLine` if there is no need to worry about truncation, or use `edit~GetLine(1, [maxLength])` if the length of the text is greater than 255.

16.14. GetText

```
>>-anEditControl~GetText---------------------------------------><
```

The GetText method returns the entire contents of the text buffer of the edit control as a string. The method works the same for both single-line and multi-line edit controls. With multi-line edit controls, the returned string will contain the carriage return and line-feed characters present in the text buffer. The programmer does not need to worry about the length of the text in the edit control. All the text is returned no matter what the length of the text is.

Return value:

The text in the edit control.

Example:

For an example, refer to the SetText example.

16.15. SetText

```
>>-anEditControl~SetText(--text--)----------------------------><
```

The SetText method places *text* in the edit control. Any current text content in the edit control is completely replaced.

Arguments:

The only argument is:

text

The text to be displayed in the edit control. For multi-line edit controls the text must contain the carriage return and line-feed characters.

Return value:

The return values are:

0

No error.

less than 0

Error. The value is the negated Operating System Error code. The absolute value of the return can be used to look up the error reason in the Windows documentation.

Example:

The following code snippets use the `GetText` and `SetText` methods in a dialog that could be used as a simple file editor:

```
  ...

::method defineDialog

  ...

  self~addEntryLine(110, , 5, 5, 170, 150, "VSCROLL HSCROLL MULTILINE")

  self~addButton(115,  5, 160, 35, 15, "Open", onOpen)
  self~addButton(116, 45, 160, 35, 15, "Save", onSave)

  ...

::method initDialog
  expose editControl
  editControl = self~getEditControl(110)

::method onOpen
  expose editControl

  if .nil == editControl then return

  fileName = self~getFile
  if fileName == "" then return

  fObj = .stream~new(fileName)
  fObj~open
  if fObj~state \== 'READY' then return

  text = fObj~charin(1, fObj~chars)
  fObj~close

  editControl~setText(text)
  editControl~setModified(.false)

::method onSave
  expose editControl

  if .nil == editControl then return

  if editControl~isModified then do
    text = editControl~getText
    self~saveFile(text)
    editControl~setModified(.false)
  end
  else do
    j = InfoDialog("There is nothing to save")
  end
```

16.16. ReplaceSelText

```
>>-anEditControl~ReplaceSelText(--text--)---------------------><
```

The ReplaceSelText method replaces the selected text in the associated edit control with a new one.

Arguments:

The only argument is:

text

The text string that is to replace the currently selected text.

Example:

```
edit = MyDialog~GetEditControl("TEXT")
if edit == .Nil then return
edit~Title = "Object Rexx is a hybrid language."
edit~Select(17,25)
    edit~ReplaceSelText("n interpreted")
    say edit~Title

Result: Object Rexx is an interpreted language.
```

16.17. SetLimit

```
>>-anEditControl~SetLimit(--chars--)---------------------------><
```

The SetLimit method sets the maximum numbers of characters that the associated edit control can contain.

Arguments:

The only argument is:

chars

The number of characters that the edit control can contain.

16.18. PasswordChar=

```
>>-anEditControl~PasswordChar=--char---------------------------><
```

The PasswordChar= method sets the character that is displayed in an edit control for which the PASSWORD or ES_PASSWORD flag is set.

Arguments:

The only argument is:

char

The character that is displayed for the typed characters.

Example:

The following example ensures that if the PASSWORD style was chosen for the edit control, the dollar sign ($) is displayed for each character typed in the edit control:

```
edit = MyDialog~GetEditControl("TEXT")
if edit == .Nil then return
edit~PasswordChar = "$"
say "The new password character is" edit~PasswordChar
```

16.19. PasswordChar

```
>>-anEditControl~PasswordChar---------------------------------><
```

The PasswordChar method retrieves the character that is displayed in an edit control for which the PASSWORD option or ES_PASSWORD style was set.

Return value:

The character that is displayed instead of the characters contained in the edit control. If the edit control is no password field or no password character was set, an empty string is returned.

Example:

For an example, refer to PasswordChar=.

16.20. SetReadOnly

```
                                +-1-+
>>-anEditControl~SetReadOnly(--+---+--)------------------------><
                                +-0-+
```

The SetReadOnly method sets or unsets the read-only flag for the associated edit control. If the read-only flag is set, the user can no longer modify the text of the edit control.

Arguments:

The only argument is:

bool

1 if the edit control is to be marked as a read-only field (the default), or 0 if new text can be typed into the edit control.

Return value:

0 if this method was successful.

16.21. SetMargins

```
>>-anEditControl~SetMargins(--left--,--right--)---------------><
```

The SetMargins method sets the left and right margins for the associated edit control. The margins determine the spacing to the left and right of the edit control.

Arguments:

The arguments are:

left

The left margin, specified in screen pixels.

right

The right margin, specified in screen pixels.

Example:

The following example sets the margins for edit control TEXT such that the left indent is 10 screen pixels and on the right there is a spacing of 5 pixels between the text and the frame of the edit control:

```
edit = MyDialog~GetEditControl("TEXT")
if edit == .Nil then return
edit~SetMargins(10, 5)
     parse value edit~Margins with left right
     say "The new left margin is" left" and the new right margin is" right
```

16.22. Margins

```
>>-anEditControl~Margins---------------------------------------><
```

The Margins method retrieves the left and right margins of the associated edit control.

Return value:

The left and right margins, in screen pixels, separated by a blank.

Example:

For an example, refer to SetMargins.

16.23. SetCue

```
>>-anEditControl~SetCue(--text--)----------------------------><
```

This method sets the "cue" text for an edit control. Cue text provides a visual hint to the user as to the purpose of the edit control. When the dialog is first shown to the user the cue text will be displayed, (provided there is no text in the edit control's buffer.)

The cue text is colored a light grey to distinguish it from regular text. When the edit control gains the focus, Windows dismisses the cue text automatically. When the control loses focus, Windows will again display the cue text, if the entry control is empty. Once the user has entered text in the edit control, the cue is no longer displayed. If the user deletes the text, then the cue is once again displayed when the edit control does not have the focus.

Note: Only single-line edit controls will display cue text. You can not use cue text with a multi-line edit control.

Operating System Version

This method is only available on Windows XP or later operating systems. If the method is used on an unsupported operating system, -4 is returned.

Arguments:

The only argument is:

text

The text to be displayed to the user as a cue. The maximum length of the text string is 255 characters.

Return value:

The return value will be one of the following:

less than -4

The value is the negated Operating System Error code. The absolute value of the return can be used to look up the error reason in the Windows documentation.

-4

The operating system does not support this method. Windows XP or later is required.

-3

The text argument is too long for this method, or there is an (internal) problem converting the text string to Unicode.

-2

There is an (internal) problem with the dialog or the dialog handle.

-1

There is an (internal) problem with the resource ID or window handle of the edit control.

0

The method succeeded.

1

The method failed.

Example:

For an example see the ShowBalloon example.

16.24. ShowBalloon

```
>>-anEditControl~ShowBalloon(--title,--text--+----------+--)--->&lt;
                                             +-,--icon--+
```

Displays a balloon information window. Balloon windows are similar in appearance to the balloon captions used in comic books to display the dialog of the characters. For an edit control the balloon window looks like it is attached to the control.

The balloon is displayed for 10 seconds and then goes away. The programmer can also dismiss the balloon sooner using the HideBalloon method. Typically balloon windows are used to convey information to the user without having to put up a message box that the user then needs to dismiss manually.

Operating System Version

This method is only available on Windows XP or later operating systems. If the method is used on an unsupported operating system, -4 is returned.

Arguments:

The arguments are:

title

A title for the balloon window. Titles are limited to 99 characters total in length.

text

The text of the balloon window. This text is limited to 1024 characters in length. The programmer can use line break characters to control the formatting of the text.

icon

Used to specify which icon is displayed on the title line. The default is the informational icon. The following keywords can be used to specify another icon, or no icon. Only the first letter of the keyword is needed and case is insignificant:

ERROR

The error icon is used.

WARN

The warning icon is used.

NONE

No icon is displayed.

Return value:

The return value will be one of the following:

less than -4

The value is the negated Operating System Error code. The absolute value of the return can be used to look up the error reason in the Windows documentation.

-4

The operating system does not support this method. Windows XP or later is required.

-3

The title or text arguments are too long for this method, or there is an (internal) problem converting one of the strings to Unicode.

-2

There is an (internal) problem with the dialog or the dialog handle.

-1

There is an (internal) problem with the resource ID or window handle of the edit control.

0

The method succeeded.

1

The method failed.

Example:

The following code snippet could come from a fictitious inventory program. An edit control is used to enter a product ID. If the user clicks on the edit control a ballon tip is shown displaying extended product information. (In a real application, the extended information might be based on what the user had entered in the edit control.)

Text is set in the edit control to give the user a 'cue' that she can click on the edit control for extra information. When the ballon tip is shown, it will automatically be dismissed after 10 seconds. This application connects a notification for the lost focus event. When the edit control loses focus, the ballon tip is hidden by the program.

```
  ...

::method defineDialog

  ...

  self~addText(10, 125, 150, 10, "Enter product ID:", "", 200)
  self~addEntryLine(IDC_ENTRYLINE, "cEntry", 10, 135, 150, 10, "AUTOSCROLLH")

  ...

::method initDialog
  expose editControl hwndEditControl

  editControl = self~getEditControl(IDC_ENTRYLINE)
  hwndEditControl = self~getItem(IDC_ENTRYLINE)

  self~connectEditNotify(IDC_ENTRYLINE, GOTFOCUS, "onFocus")
  self~connectEditNotify(IDC_ENTRYLINE, LOSTFOCUS, "onLostFocus")

  ret = editControl~setCue("Click here for extended product information")
  if ret <> 0 then ErrorDialog( "Set cue error return:" ret )

  -- Capture the mouse activate messages: WM_MOUSEACTIVATE message == 0x0021
  self~addUserMsg(onMouseActivate, "0x0021", "0xFFFFFFFF", 0, 0, 0, 0)

  ...

::method onMouseActivate
  expose editControl hwndEditControl

  if self~getFocus == hwndEditControl then do
    title = "Extended Product Information"
    text = "Product ID:"   || "9"x    || "4538-D32"       || .endOfLine || -
```

```
            "In stock:"     || "9"x    || "789"            || .endOfLine || -
            "Back ordered:" || "9"x    || "No"             || .endOfLine || -
            "EOL:"          || "0909"x || "January 2008"   || .endOfLine || -
            "Client:"       || "0909"x || "AT&T"           || .endOfLine || .endOfLine -

            "Notes: This product has served a limited number of customers" -
            "and should be considered archic.  It will be replaced by a"   -
            "superior product line."

    ret = editControl~showBalloon(title, text, "e");
    if ret <> 0 then ErrorDialog("Show balloon error return:" ret)
  end

::method onLostFocus
  expose editControl

  -- When the edit control loses the focus, dismiss the alloon.
  ret = editControl~hideBalloon
  if ret <> 0 then ErrorDialog("Hide balloon error return:" ret)
```

16.25. HideBalloon

```
>>-anEditControl~HideBalloon----------------------------------><
```

Hides the balloon information window displayed by the ShowBalloon method. The operating system will dismiss the balloon window automatically after 10 seconds with no programmer intervention. This method can is used to dismiss the ballon sooner.

Operating System Version

This method is only available on Windows XP or later operating systems. If the method is used on an unsupported operating system, -4 is returned.

Arguments:

This method takes no arguments.

Return value:

The retrun value will be one of the following:

less than -4

The value is the negated Operating System Error code. The absolute value of the return can be used to look up the error reason in the Windows documentation.

-4

The operating system does not support this method. Windows XP or later is required.

-2

There is an (internal) problem with the dialog or the dialog handle.

-1

There is an (internal) problem with the resource ID or window handle of the edit control.

0

The method succeeded.

1

The method failed.

Example:

For an example see the ShowBalloon example.

16.26. AddStyle

```
                              +----------------+
                              V                |
>>-anEditControl~AddStyle(--"---+-UPPER------+-+--"--)---------><
                                +-LOWER------+
                                +-NUMBER-----+
                                +-WANTRETURN-+
                                +-OEM--------+
                                +-TAB--------+
                                +-GROUP------+
```

Adds one or more edit control styles to the edit control. This method works with the window styles that can be changed by accessing the edit control window directly after the edit control has been created.

Arguments:

The style argument can be one or more of the following keywords. (Each keyword is separated by a blank.):

UPPER

Converts all characters to uppercase as they are typed into the edit control.

LOWER

Converts all characters to lowercase as they are typed into the edit control.

NUMBER

Only allows digits to be typed into the edit control. If the user types an non-digit character an information balloon is displayed informing the user that only digits are allowd. This style does not prevent the user from pasting non-digits into the control.

WANTRETURN

This keyword only effects multi-line edit controls. With this style when a user hits the enter key the line break characters are inserted into the text buffer. Without the style, when the user hits enter the default push button of the dialog is pressed.

OEM

Converts text entered in the edit control from the Windows character set to the OEM character set and then back again to the Windows character set. This ensures proper character conversion in certain circumstances. Microsoft states that this style is most useful when edit controls contain file names that will be used with file systems that do not support Unicode. For further details see the Windows documentation provided by Microsoft.

TAB

Adds the tabstop tab stop style.

GROUP

Adds the group style.

Return value:

Negative values indicate error, non-negative values indicate success.

-4 or less

The value is the negated Operating System Error code. The absolute value of the return can be used to look up the error reason in the Windows documentation.

-3

The was no valid keyword in the style argument. Note that if the style argument contains at least one valid keyword other invalid keywords are then ignored.

-2

There is an (internal) problem with the dialog or the dialog handle.

-1

There is an (internal) problem with the resource ID or window handle of the edit control.

other

The numeric value of the edit control's previous style.

Example:

This example shows the syntax to add the number, tabstop, and group styles to an edit control.

```
editControl = dlg~getEditControl(IDC_ENTRYLINE)
if .nil == editControl  then return
editControl~addStyle("NUMBER TAB GROUP")
```

16.27. RemoveStyle

```
                                  +----------------+
                                  V                |
>>-anEditControl~RemoveStyle(--"---+-UPPER------+-+--"--)------><
                                    +-LOWER------+
                                    +-NUMBER-----+
                                    +-WANTRETURN-+
                                    +-OEM--------+
                                    +-TAB--------+
                                    +-GROUP------+
```

Remove one or more edit control styles from the edit control. This method works with the window styles that can be changed by accessing the edit control window directly after the edit control has been created.

Arguments:

The style argument can be one or more of the style keywords. (Each keyword is separated by a blank.) See the argument list for the `AddStyle` method for detals on the keywords.

Return value:

Negative values indicate error, non-negative values indicate success.

-4 or less

The value is the negated Operating System Error code. The absolute value of the return can be used to look up the error reason in the Windows documentation.

-3

The was no valid keyword in the style argument. Note that if the style argument contains at least one valid keyword other invalid keywords are then ignored.

-2

There is an (internal) problem with the dialog or the dialog handle.

-1

There is an (internal) problem with the resource ID or window handle of the edit control.

other

The numeric value of the edit control's previous style.

Example:

This example shows the syntax to remove the lower and the tabstop styles from an edit control.

```
editControl = dlg~getEditControl(IDC_ENTRYLINE)
if .nil == editControl  then return
editControl~removeStyle("LOWER GROUP")
```

16.28. ReplaceStyle

```
                                +---------------+     +---------------+
                                V               |     V               |
>>-anEditControl~ReplaceStyle(-"--+-UPPER-----+-+-",-"--+-UPPER-----+-+-"-)-->&lt;
                                  +-LOWER-----+         +-LOWER-----+
                                  +-NUMBER----+         +-NUMBER----+
                                  +-WANTRETURN+         +-WANTRETURN+
                                  +-OEM-------+         +-OEM-------+
                                  +-TAB-------+         +-TAB-------+
                                  +-GROUP-----+         +-GROUP-----+
```

Removes and adds edit control styles for the edit control in one operation. This method works with the window styles that can be changed by accessing the edit control window directly after the edit control has been created.

Arguments:

The two arguments can be one or more of the style keywords. (Each keyword is separated by a blank.) See the argument list for the `AddStyle` method for detals on the keywords.

remove

The first argument is the list of styles to remove.

add

The second argument is the list of styles to add.

Return value:

Negative values indicate error, non-negative values indicate success.

-4 or less

The value is the negated Operating System Error code. The absolute value of the return can be used to look up the error reason in the Windows documentation.

-3

The was no valid keyword in one or both of the style arguments. Note that if a style argument contains at least one valid keyword other invalid keywords are then ignored.

-2

There is an (internal) problem with the dialog or the dialog handle.

-1

There is an (internal) problem with the resource ID or window handle of the edit control.

other

The numeric value of the edit control's previous style.

Example:

The following would change an edit control from one that only allows numbers to one that uppercases all letters entered, while removing the tabstop style and adding the group style.

```
editControl = dlg~getEditControl(IDC_ENTRYLINE)
if .nil == editControl then return
editControl~replaceStyle("NUMBER TAB", "UPPER GROUP")
```

16.29. GetStyle

```
>>-anEditControl~GetStyle-------------------------------------------------><
```

Returns the edit control's current style as a blank delimited list of style keywords.

Arguments:

The method takes no arguments.

Return value:

A string consisting of blank delimited style keywords. The possible keywords are: VISIBLE HIDDEN TAB NOTAB DISABLED ENABLED GROUP HSCROLL VSCROLL PASSWORD MULTILINE AUTOSCROLLH AUTOSCROLLV READONLY WANTRETURN KEEPSELECTION UPPER LOWER NUMBER OEM RIGHT CENTER LEFT. The meaning of these style keywords is documented in the AddEntryLine method.

In addition, if there is an internal error, it is possible that a negative error code might be returned. These error codes are documented here, but it is unlikely that the ooDialog programmer would ever see them.

-3 or less

The value is the negated Operating System Error code. The absolute value of the return can be used to look up the error reason in the Windows documentation.

-2

There is an (internal) problem with the dialog or the dialog handle.

-1

There is an (internal) problem with the resource ID or window handle of the edit control.

Example:

The following example could be from a fictitious application that changes the style of an edit control back and forth between accepting numbers only and accepting all characters based on the options the user has selected.

```
editControl = dlg~getEditControl(IDC_ENTRYLINE)
```

```
if .nil == editControl then return
style = editControl~getStyle
if style~wordPos("NUMBER") <> 0 then
  editControl~removeStyle("NUMBER")
...
```

Chapter 17. ListControl Class

A list view control is a window that displays a collection of items, with each item consisting of an icon and a label. It provides several ways of arranging and displaying items. Refer to OODLIST.REX in the OODIALOG\SAMPLES directory for an example.

Use the GetListControl method (see page GetListControl) to retrieve an object of the ListControl class.

Notes

The ListControl class is the ooDialog interface to the Windows *List-View* control. This should not be confused with the Windows *List Box* control. The ooDialog ListBox class provides the interface to the Windows List Box control. In general, the List-View control is more powerful than the List Box control.

List-View **Extended Styles**: The List-View control has been updated by Microsoft to include a number of *extended* styles. These styles are similar to the styles that can be used to create a ListControl, (see the AddListControl method,) like the `NOHEADER` or `SINGLESEL` styles. However, the way that they are added or removed from the ListControl is different. For instance, the extended styles can not be added to a ListControl at the time of creation. They can only be added after the underlying Windows control has been created. For all practical purposes this means in the InitDialog method, or at some point in the life cycle of the dialog after that.

Some of the extended styles are only available on later versions of the Windows OS. As an example, the `LABELTIP` and `SIMPLESELECT` styles are only available on Windows XP. The documentation on the extended styles will note any requirements for each style. To work with the extended List-View styles, use the AddExtendedStyle, the RemoveExtendedStyle, or the ReplaceExtendedStyle methods.

> **Note:** There are a number of nuances to the behavior of the different extended styles on the different versions of Windows, far too many to try and detail in this documentation. The behavior documented here is the general behavior on Windows XP with service pack 2.
>
> If there is a need for detailed information concerning the subtle differences in behavior on earlier versions of Windows, the programmer should consult the Windows documentation.

Requires:

The ListControl class requires the class definition file `oodwin32.cls`:

```
::requires "oodwin32.cls"
```

Methods:

Instances of the ListControl class implement the methods listed in the ListControl Instance Methods table.

Table 17-1. ListControl Instance Methods

Method...	...on page

Method...	...on page
Add	Add
AddRow	AddRow
AddExtendedStyle	AddExtendedStyle
AddStyle	AddStyle
AlignLeft	AlignLeft
AlignTop	AlignTop
Arrange	Arrange
BkColor	BkColor
BkColor=	BkColor=
Check	Check
CheckAll	CheckAll
ColumnInfo	ColumnInfo
ColumnWidth	ColumnWidth
Delete	Delete
DeleteAll	DeleteAll
DeleteColumn	DeleteColumn
Deselect	Deselect
DropHighlighted	DropHighlighted
Edit	Edit
EndEdit	EndEdit
EnsureVisible	EnsureVisible
Find	Find
FindNearestXY	FindNearestXY
FindPartial	FindPartial
FirstVisible	FirstVisible
Focus	Focus
Focused	Focused
GetCheck	GetCheck
GetExtendedStyle	GetExtendedStyle
GetExtendedStyleRaw	GetExtendedStyleRaw
GetHoverTime	GetHoverTime
Insert	Insert
InsertColumn	InsertColumn
IsChecked	IsChecked
ItemInfo	ItemInfo
ItemPos	ItemPos
ItemState	ItemState
ItemText	ItemText

Method...	...on page
Items	Items
ItemsPerPage	ItemsPerPage
Last	Last
LastSelected	LastSelected
Modify	Modify
ModifyColumn	ModifyColumn
Next	Next
NextLeft	NextLeft
NextRight	NextRight
NextSelected	NextSelected
Prepare4nItems	Prepare4nItems
Previous	Previous
PreviousSelected	PreviousSelected
RedrawItems	RedrawItems
RemoveImages	RemoveImages
RemoveExtendedStyle	RemoveExtendedStyle
RemoveStyle	RemoveStyle
RemoveSmallImages	RemoveSmallImages
ReplaceExtendedStyle	ReplaceExtendedStyle
ReplaceStyle	ReplaceStyle
RestoreEditClass	RestoreEditClass
Scroll	Scroll
Select	Select
Selected	Selected
SelectedItems	SelectedItems
SetColumnWidth	SetColumnWidth
SetHoverTime	SetHoverTime
SetImages	SetImages
SetItemPos	SetItemPos
SetItemState	SetItemState
SetItemText	SetItemText
SetSmallImages	SetSmallImages
SmallSpacing	SmallSpacing
SnapToGrid	SnapToGrid
Spacing	Spacing
StringWidth	StringWidth
SubclassEdit	SubclassEdit
TextBkColor	TextBkColor

Method...	...on page
TextBkColor=	TextBkColor=
TextColor	TextColor
TextColor=	TextColor=
Uncheck	Uncheck
UncheckAll	UncheckAll
Update	Update
UpdateItem	UpdateItem

17.1. View Styles

List view controls can display their contents in different views. The current view is specified by the window style of the control. Additional window styles define the alignment of the items and the functionality of the list view control. The different views are:

Icon view

Each item appears as a full-sized icon with a label below it. The user can drag the items to any location in the list view.

Small-icon view

Each item appears as a small icon with a label to the left of it. The user can drag the items to any location.

List view

Each item appears as a small icon with a label to the left of it. The user cannot drag the items.

Report view

Each item appears on a separate line with information arranged in columns. The leftmost column contains the small icon and the label. All following columns contain subitems as specified by the application.

17.2. Methods of the ListControl Class

The following sections describe the individual methods.

17.2.1. ReplaceStyle

```
                                                +-------------------+
                                                V                   |
>>-aListControl~ReplaceStyle(--oldStyle--,--"----+-ICON----------+-+--"--)-><
                                                  +-SMALLICON-----+
```

```
+-LIST----------+
+-REPORT--------+
+-ALIGNLEFT-----+
+-ALIGNTOP------+
+-AUTOARRANGE---+
+-ASCENDING-----+
+-DESCENDING----+
+-EDIT----------+
+-HSCROLL-------+
+-VSCROLL-------+
+-NOSCROLL------+
+-NOHEADER------+
+-NOSORTHEADER--+
+-NOWRAP--------+
+-SINGLESEL-----+
+-SHOWSELALWAYS-+
+-SHAREIMAGES---+
+-GROUP---------+
+-HIDDEN--------+
+-NOTAB---------+
+-NOBORDER------+
```

The ReplaceStyle method removes a window style of a list view control and sets new styles.

Arguments:

The arguments are:

oldStyle

The window style to be removed.

newStyle

The new window styles to be set, which is one or more of the styles listed in the syntax diagram, separated by blanks. For an explanation of the different styles, refer to AddListControl.

Return value:

0 if this method fails.

Example:

The following example replaces a small-icon list with an icon list and connects the new bitmap file:

```
::method Icon
  curList = self~GetListControl(104)
  curList~SetImages("ilist.bmp",16,12)
  curList~ReplaceStyle("SMALLICON","ICON")
```

17.2.2. AddStyle

```
                              +-------------------+
                              V                   |
>>-aListControl~AddStyle(--"----+-ICON----------+-+--"--)------><
                                +-SMALLICON-----+
                                +-LIST----------+
                                +-REPORT--------+
                                +-ALIGNLEFT-----+
                                +-ALIGNTOP------+
                                +-AUTOARRANGE---+
                                +-ASCENDING-----+
                                +-DESCENDING----+
                                +-EDIT----------+
                                +-HSCROLL-------+
                                +-VSCROLL-------+
                                +-NOSCROLL------+
                                +-NOHEADER------+
                                +-NOSORTHEADER--+
                                +-NOWRAP--------+
                                +-SINGLESEL-----+
                                +-SHOWSELALWAYS-+
                                +-SHAREIMAGES---+
                                +-GROUP---------+
                                +-HIDDEN--------+
                                +-NOTAB---------+
                                +-NOBORDER------+
```

The AddStyle method adds new window styles to a list view control.

Arguments:

The only argument is:

style

The window styles to be added, which is one or more of the styles listed in the syntax diagram, separated by blanks. For an explanation of the different styles, refer to AddListControl.

Return value:

0 if this method fails.

17.2.3. RemoveStyle

```
                                +-------------------+
                                V                   |
>>-aListControl~RemoveStyle(--"----+-ICON----------+-+--"--)---><
                                  +-SMALLICON-----+
                                  +-LIST----------+
                                  +-REPORT--------+
```

```
+-ALIGNLEFT-----+
+-ALIGNTOP------+
+-AUTOARRANGE---+
+-ASCENDING-----+
+-DESCENDING----+
+-EDIT----------+
+-HSCROLL-------+
+-VSCROLL-------+
+-NOSCROLL------+
+-NOHEADER------+
+-NOSORTHEADER--+
+-NOWRAP--------+
+-SINGLESEL-----+
+-SHOWSELALWAYS-+
+-SHAREIMAGES---+
+-GROUP---------+
+-HIDDEN--------+
+-NOTAB---------+
+-NOBORDER------+
```

The RemoveStyle method removes one or more window styles of a list view control.

Arguments:

The only argument is:

style

The window styles to be removed, which is one or more of the styles listed in the syntax diagram, separated by blanks. For an explanation of the different styles, refer to AddListControl.

Return value:

0 if this method fails.

17.2.4. AddExtendedStyle

```
                                    +--------------------+
                                    V                    |
>>-aListControl~AddExtendedStyle(--"---+-BORDERSELECT-----+--"--)------------><
                                      +-CHECKBOXES-------+
                                      +-DOUBLEBUFFER-----+
                                      +-FLATSB-----------+
                                      +-FULLROWSELECT----+
                                      +-GRIDLINES--------+
                                      +-HEADERDRAGDROP---+
                                      +-INFOTIP----------+
                                      +-LABELTIP---------+
                                      +-MULTIWORKAREAS---+
```

```
+-ONECLICKACTIVATE-+
+-REGIONAL---------+
+-SIMPLESELECT-----+
+-SUBITEMIMAGES----+
+-TRACKSELECT------+
+-TWOCLICKACTIVATE-+
+-UNDERLINECOLD----+
+-UNDERLINEHOT-----+
```

The `AddExtendedStyle` method adds one or more of the extended List-View styles to a ListControl. These styles can not be added until the underlying Windows List-View control has been created. They can be added in the `InitDialog` method or any time thereafter.

Arguments:

The only argument is:

style

The extended styles to be added. This argument is a list containing one or more of the following style keywords, separated by blanks. Invalid words in the list are ignored. If there are no valid keywords at all in the list, then an error is returned.

BORDERSELECT

When an item is selected the border color of the item changes rather than the item being highlighted.

CHECKBOXES

Check boxes are visible and functional in all list view modes except tile mode.

DOUBLEBUFFER

Windows XP or later only. The list-view control is painted using double buffering. This reduces flicker.

FLATSB

Enables flat scroll bars.

FULLROWSELECT

Report view only. When a row is selected, the whole row is highlighted rather than just the first column.

GRIDLINES

Report view only. Gridlines are drawn around all items and sub-items. This gives a spreadsheet-like appearance to the list view control.

HEADERDRAGDROP

Report view only. The user can drag and drop the headers to rearrange the columns.

INFOTIP

With this style a notification is sent to the parent window before displaying an item's tooltip. (The current version of ooDialog can not take advantage of this style. It is planned for a future enhancement.)

LABELTIP

Windows XP or later only. In any list view mode, if an item's label is only partially visible, the complete label will be displayed in a fashion similar to a tool tip.

MULTIWORKAREAS

The current version of ooDialog can not take advantage of this style. It is planned for a future enhancement.

ONECLICKACTIVATE

Enables the list-view control to send an item activate notification when the user clicks one time on an item. It also enables hot tracking in the list-view control. Both the underline cold and the underline hot styles need either one click activate or two click activate styles to be enabled.

REGIONAL

Icon view only. Sets the window region to only include an item's icon and text.

SIMPLESELECT

Windows XP and later only. Icon view only. Moves the state image to the top right of the icon image. If the user changes the state using the space bar, all the selected items cycle, not just the item with the focus.

SUBITEMIMAGES

Report view only. Allows images to be displayed for subitems.

TRACKSELECT

Enables hot track selection. The user can select an item by leaving the mouse over an item for a certain period of time, the "hover" time. This period of time can be changed from the default by using the SetHoverTime method.

TWOCLICKACTIVATE

Enables the list-view control to send an item activate notification when the user clicks on an item after it has been selected. When the user clicks on an item it will be selected. At that point, if the user clicks on the item a second time, the item will be activated. Note that this is different from double-clicking to activate. Any amount of time can pass between the click that selects the item and the click that activates the item. (Provided the item remains selected of course.)

It also works with track select. If track select is enabled, the user can select an item by pausing the mouse over the item. At that point, if the user then clicks on the item it will be activated.

UNDERLINECOLD

Underlining gives the user a visual clue that a single click on an item will activate it. Underlining requires either `ONECLICKACTIVATE` or `TWOCLICKACTIVE` to also be enabled. An item becomes hot when the mouse is over it. Underline cold underlines all the cold items when a single click will activate them. When one click activate is enabled, this would then be all items. When two click activate is enabled this would then be only those items that were selected.

UNDERLINEHOT

The underline hot style is similar to the underline cold style. See that style above for the explanation of some of the terms. With underline hot, the item is underlined when the mouse is over it, if one click will activate the item.

Return value:

The return values are:

0

Success.

-1

An (internal) problem with the control's window handle.

-3

The style argument did not contain any of the extended style keywords.

Example:

The following example shows how to initially add the extended styles to a list view control. This is done in the `InitDialog` method, which executes after the underlying Windows dialog has been created, but before it is displayed on the screen. This is the earliest point in time that the extended style methods can be used. The list view is in the report view and will have check boxes and grid lines. The user will be able to use drag and drop to rearrange the columns. When a row is selected, the full row will be highlighted rather than just the first column.

```
::method initDialog
  expose listView

  ...
  listView = self~getListControl(IDC_LIST)
  if listView == .nil then return

  listView~addExtendedStyle("CHECKBOXES GRIDLINES FULLROWSELECT HEADERDRAGDROP")
  ...
```

17.2.5. RemoveExtendedStyle

```
                                        +--------------------+
                                        V                    |
>>-aListControl~RemoveExtendedStyle(--"----+-BORDERSELECT-----+--"--)-------><
                                          +-CHECKBOXES-------+
                                          +-DOUBLEBUFFER-----+
                                          +-FLATSB-----------+
                                          +-FULLROWSELECT----+
                                          +-GRIDLINES--------+
                                          +-HEADERDRAGDROP---+
                                          +-INFOTIP----------+
                                          +-LABELTIP---------+
                                          +-MULTIWORKAREAS---+
                                          +-ONECLICKACTIVATE-+
                                          +-REGIONAL---------+
                                          +-SIMPLESELECT-----+
                                          +-SUBITEMIMAGES----+
                                          +-TRACKSELECT------+
                                          +-TWOCLICKACTIVATE-+
                                          +-UNDERLINECOLD----+
                                          +-UNDERLINEHOT-----+
```

The `RemoveExtendedStyle` method removes one or more extended styles of a List-View control.

Arguments:

The only argument is:

style

The extended style(s) to be removed. This is one or more of the styles listed in the syntax diagram, separated by blanks. For an explanation of the different styles, refer to the AddExtendedStyle method.

Return value:

The return values are:

0

Success.

-1

An (internal) problem with the control's window handle.

-3

The style argument did not contain any of the extended style keywords.

Example:

The following example removes the check box style from the list view.

```
listView = self~getListControl(IDC_LIST)
if listView == .nil then return

listView~removeExtendedStyle("CHECKBOXES")
```

17.2.6. ReplaceExtendedStyle

```
                                                      +--------------------+
                                                      V                    |
>>-aListControl~ReplaceExtendedStyle(-oldStyle--,--"---+-BORDERSELECT-----+--"-)-><
                                                        +-CHECKBOXES-------+
                                                        +-DOUBLEBUFFER-----+
                                                        +-FLATSB-----------+
                                                        +-FULLROWSELECT----+
                                                        +-GRIDLINES--------+
                                                        +-HEADERDRAGDROP---+
                                                        +-INFOTIP----------+
                                                        +-LABELTIP---------+
                                                        +-MULTIWORKAREAS---+
                                                        +-ONECLICKACTIVATE-+
                                                        +-REGIONAL---------+
                                                        +-SIMPLESELECT-----+
                                                        +-SUBITEMIMAGES----+
                                                        +-TRACKSELECT------+
                                                        +-TWOCLICKACTIVATE-+
                                                        +-UNDERLINECOLD----+
                                                        +-UNDERLINEHOT-----+
```

The `ReplaceExtendedStyle` method removes some of the List-View control's extended styles and adds new extended styles. This is the equivalent of first using the `RemoveExtendeStyle` method and then using the `AddExtendedStyle` method.

Arguments:

The arguments are:

oldStyle

The extended style(s) to be removed. This is one or more of the styles listed in the syntax diagram, separated by blanks. For an explanation of the different styles, refer to the AddExtendedStyle method.

newStyle

The extended style(s) to be added. Again, this is one or more of the styles listed in the syntax diagram, separated by blanks. See the AddExtendedStyle method for the style descriptions.

Return value:

The return values are:

0

Success.

-1

An (internal) problem with the control's window handle.

-3

One, or both, of the style arguments had none of the extended style keywords.

Example:

The following example removes the cold and hot underlining and adds check boxes to the list view.

```
listView = self~getListControl(IDC_LIST)
if listView == .nil then return

removeStyle = "UNDERLINECOLD UNDERLINEHOT"
addStyle = "CHECKBOXES"

listView~replaceExtendedStyle(removeStyle, addStyle)
```

17.2.7. GetExtendedStyle

```
>>-aListControl~GetExtendedStyle--------------------------------------------><
```

The `GetExtendedStyle` method returns the extended styles of a `ListControl`. The styles are returned as a string of blank delimited style keywords. These are the same keywords as those used to set the styles.

Arguments:

This method takes no arguments.

Return value:

The returned string can contain one or more of the following style keywords. See the AddExtendedStyle method for the style descriptions:

- BORDERSELECT
- CHECKBOXES
- DOUBLEBUFFER
- FLATSB
- FULLROWSELECT
- GRIDLINES
- HEADERDRAGDROP

- INFOTIP
- LABELTIP
- MULTIWORKAREAS
- ONECLICKACTIVATE
- REGIONAL
- SIMPLESELECT
- SUBITEMIMAGES
- TRACKSELECT
- TWOCLICKACTIVATE
- UNDERLINECOLD
- UNDERLINEHOT

Example:

The following example comes from a fictious application where the user can elect to use the track select style, or not. With this style, an item can be selected by pausing the mouse over it for a period of time specified by the hover time. The application also allows the user to increase or decrease the hover time. In this example when the methods to increment or decrement the hover time are invoked, the application first checks to see if the list view control has the track select style. Then the change is only made if the list view control does have the style.

```
::method initDialog
  expose listView

  ...
  listView = self~getListControl(IDC_LIST)
  if listView == .nil then return

  listView~addExtendedStyle("CHECKBOXES FULLROWSELECT TRACKSELECT")
  ...

::method onCheckboxClick
  expose listView

  chkBox = self~getCheckControl(IDC_CHECK_USETRACKSELECT)
  if chkBox~isChecked == "CHECKED" then
    listView~addExtendedStyle("TRACKSELECT")
  else
    listView~removeExtendedStyle("TRACKSELECT")

::method onDecrement
  expose listView

  styles = listView~getExtendedStyle
  if styles~wordpos("TRACKSELECT") == 0 then return
```

```
    time = listView~getHoverTime - 100
    if time > 0 then listView~setHoverTime(time)

  ::method onIncrement
    expose listView

    styles = listView~getExtendedStyle
    if styles~wordpos("TRACKSELECT") == 0 then return

    -- Let the user increase the hover time to any length they want.
    listView~setHoverTime(list~getHoverTime + 100)
```

17.2.8. GetExtendedStyleRaw

```
>>-aListControl~GetExtendedStyleRaw-----------------------------------------><
```

The `GetExtendedStyleRaw` method returns the extended styles of a `ListControl` as the numeric value of the extended style flags. This method is provided for programmers familiar with the Windows API who may wish to know, or to work with, the actual value of the flags.

Arguments:

This method takes no arguments.

Return value:

The returned value is the numeric value of the extended style flags currently in effect for the list-view control.

Example:

The following example is the same example as the GetExtendedStyle example. It simply replaces the `GetExtendedStyle` invocation with a `GetExtendedStyleRaw` invocation.

```
  ::method initDialog
    expose listView

    ...
    listView = self~getListControl(IDC_LIST)
    if listView == .nil then return

    listView~addExtendedStyle("CHECKBOXES FULLROWSELECT TRACKSELECT")
    ...

  ::method onCheckboxClick
    expose listView

    chkBox = self~getCheckControl(IDC_CHECK_USETRACKSELECT)
    if chkBox~isChecked == "CHECKED" then
      listView~addExtendedStyle("TRACKSELECT")
    else
```

```
    listView~removeExtendedStyle("TRACKSELECT")

::method onDecrement
  expose listView

  if binaryAnd(listView~getExtendedStyleRaw, "0x00000008") <> 0 then return

  time = listView~getHoverTime - 100
  if time > 0 then listView~setHoverTime(time)

::method onIncrement
  expose listView

  if binaryAnd(listView~getExtendedStyleRaw, "0x00000008") <> 0 then return

  -- Let the user increase the hover time to any length they want.
  listView~setHoverTime(list~getHoverTime + 100)
```

17.2.9. GetHoverTime

```
>>-aListControl~GetHoverTime----------------------------------><
```

This method retrieves the current hover time. The hover time is expressed as milliseconds. The hover time only affects list view controls that have the track select, one click activate, or two click activate extended list view styles. See the AddExtendedStyle for a description of these styles.

Arguments:

This method takes no arguments

Return value:

The hover time. A value of -1 indicates the hover time is the system default hover time.

Example:

The following example comes from a program where the one click activate and track select styles are enabled. Orignally the hover time is set to .4 seconds. However the application allows the users to increase or decrease this time to match their preference. A menu item (or a button could be used) is connected to the `onIncrement` and `onDecrement` methods. When either of the methods is invoked, the current hover time is examined and then either incremented or decremented by .1 of a second.

```
::method initDilaog
  expose list

  ...
  list = self~getListControl(IDC_LIST)
  list~addExtendedStyle("ONECLICKACTIVATE TRACKSELECT FULLROWSELECT")
  list~setHoverTime(400)
  ...
```

```
::method onDecrement
  expose list

  time = list~getHoverTime - 100
  if time > 0 then list~setHoverTime(time)

::method onIncrement
  expose list

  -- Let the user increase the hover time to any length they want.
  list~setHoverTime(list~getHoverTime + 100)
```

17.2.10. SetHoverTime

```
>>-aListControl~SetHoverTime(--+------+--)--------------------><
                               +-time-+
```

This method is used to change the *hover* time. The hover time only affects list view controls that have the track select, one click activate, or two click activate extended list view styles. See the AddExtendedStyle for a description of these styles.

Arguments:

The only optional argument is:

time

Specify the hover time in milliseconds. If this argument is omitted, the hover time is set to the system default. A negative number will also set the hover time back to the system defualt.

Return value:

The previous hover time. A value of -1 indicates the previous hover time was the system default.

Example:

The following example adds the one click activate, the track select, and the full row select extended list view styles to a `ListControl`. The hover time is set to .4 seconds. If the user pauses his mouse over a row in the list view for .4 of a second the row is automatically selectd.

```
::method initDilaog

  ...
  list = self~getListControl(IDC_LIST)
  list~addExtendedStyle("ONECLICKACTIVATE FULLROWSELECT TRACKSELECT")

  oldTime = list~setHoverTime(400)
  ...
```

17.2.11. Check

```
>>-aListControl~Check(--index--)-------------------------------------------><
```

The `Check` method sets the check mark for a list view item. This method is only valid for list view controls that have the check boxes extended style. (See the AddExtendedStyle method for a description of the check boxes extended style.)

Arguments:

The only argument is:

index

The index of the item for which to set the check mark.

Return value:

The return values are:

0

Success.

-1

An (internal) problem with the control's window handle.

-2

The list view control does not have the check boxes extended style.

-3

The index is invalid.

Example:

The following example is from a mailing list application. The user selects which entries to include in any mailing. All items in a list view that have their check boxes checked are included in the mailing. The application has a feature that allows the user to automatically select all items with a specified zip code.

```
::method onUseZip

  text = self~getEditControl(IDC_EDIT_ZIPCODE)~getText
  if self~validateZip(text) then do
     list = self~getListControl(IDC_LIST)
     do i = 0 to addresses~items - 1
       infoTable = address[i + 1]
       if infoTable['zip'] == text then list~check(i)
     end
  end
```

17.2.12. Uncheck

```
>>-aListControl~Uncheck(--index--)----------------------------------------><
```

The `Uncheck` method removes the check mark for a list view item. This method is only valid for list view controls that have the check boxes extended style. (See the AddExtendedStyle method for a description of the check boxes extended style.)

Arguments:

The only argument is:

index

The index of the item where the check box check is to be removed.

Return value:

The return values are:

0

Success.

-1

An (internal) problem with the control's window handle.

-2

The list view control does not have the check boxes extended style.

-3

The index is invalid.

Example:

The following example is from a doctor's office custom accounting application. A list box is filled with information for every patient. One button in the application checks every patient with an outstanding balance, they will be mailed a statement for that balance. The doctor does not believe in sending billing statements to patients that have a very low balance. However what a "low" balance is depends on how he is feeling at statement time. Therefore the application allows the office manager to dynamically remove some patients from the statement list.

```
::method onDeferBilling
  expose billingList

  text = self~getEditControl(IDC_EDIT_MINAMOUNT)~getText
  minimun = self~decimalNumber(text)

  if minimum > 0 then do i = 0 to billingList~items
    if billingList~isChecked(i) then do

      -- Column 6 in the list is the patient's current balance.
```

```
      owes = billingList~itemText(i, 6)
      if self~decimalNumber(owes) <= minimum then billingList~uncheck(i)

    end
  end
```

17.2.13. CheckAll

```
>>-aListControl~CheckAll-------------------------------------------------><
```

The `CheckAll` method sets the check mark for every list view item. This method is only valid for list view controls that have the check boxes extended style. (See the AddExtendedStyle method for a description of the check boxes extended style.)

Arguments:

This method takes no arguments.

Return value:

The return values are:

0

Success.

-1

An (internal) problem with the control's window handle.

-2

The list view control does not have the check boxes extended style.

Example:

The following example is from a program that has a push button that allows the user to check all the check boxes:

```
::method onCheckAll

  list = self~getListControl(IDC_LIST_REPUBLICANS)
  list~checkAll
```

17.2.14. UncheckAll

```
>>-aListControl~UncheckAll-----------------------------------------------><
```

The `UncheckAll` method clears the check mark for every list view item. This method is only valid for list view controls that have the check boxes extended style. (See the AddExtendedStyle method for a description of the check boxes extended style.)

Arguments:

This method takes no arguments.

Return value:

The return values are:

0

Success.

-1

An (internal) problem with the control's window handle.

-2

The list view control does not have the check boxes extended style.

Example:

The following example is from a program that has a push button that allows the user to remove the check mark from all the check boxes:

```
::method onUncheckAll

  list = self~getListControl(IDC_LIST_REPUBLICANS)
  list~uncheckAll
```

17.2.15. GetCheck

```
>>-aListControl~GetCheck(--index--)----------------------------------------><
```

The `GetCheck` method determines if the item at the specified list index has a checked check box. This method will return a code indicating that the list view control does not have the check boxes extended style, if appropriate. (See the AddExtendedStyle method for a description of the check boxes extended style.)

Arguments:

The only argument is:

index

The index of the item to query.

Return value:

The return values are:

1

The item has a check box that is checked.

0

The item has a check box, and it is not checked.

-1

An (internal) problem with the control's window handle.

-2

The list view control does not have the check boxes extended style.

-3

The index is invalid.

Example:

The following example could be used as a shortcut method to determine if a list view control currently has the check box extended style.

```
...
list = self~getListControl(IDC_LV_DOCTORS)
if list~getCheck(0) == -2 then
  return   -- The list view does not currently have the check box style in effect

do i = 0 to list~items - 1
  ...
end
```

17.2.16. IsChecked

```
>>-aListControl~IsChecked(--index--)----------------------------------------><
```

The `IsChecked` method determines if a check box for a list view item is checked or not. This method will always return true or false. It can be used if the list view control has or does not have the check boxes extended style. (See theAddExtendedStyle method for a description of the check boxes extended style.)

Arguments:

The only argument is:

index

The index of the item to query.

Return value:

The return values are:

true

The list view item has a check box and it is checked.

false

There is no check mark for the list view item. Either the list view has check boxes and the check box is not checked, the index is out of range or not valid, or the list view does not have the check box extended style.

Example:

The following example is from a doctor's office custom accounting application. A list box is filled with information for every patient. After the office mananger has finished working with the accounts a mail merge is done to produce statements for all patients that are to receive statements for the month. For every patient item in the list view, if the item has a check mark, then that patient is mailed a statement.

```
::method onCreateMailMerge
  expose patientList

  do i = 0 to patientList~items
    if billingList~isChecked(i) then do
      self~addToMailMerge(billingList, i)
    end
  end
```

17.2.17. InsertColumn

```
                              +-0------+
>>-aListControl~InsertColumn(--+--------+--,--text--,-->
                              +-column-+

>--,--width--+---------------------+--)---------------><
             |       +-LEFT---+    |
             +-,--"--+-CENTER-+--"-+
                     +-RIGHT--+
```

The InsertColumn method sets the attributes of a report list view column.

Arguments:

The arguments are:

column

The number of the column. 0 is the first column and the default.

text

The text of the column heading.

width

The width of the column, in pixels.

align

The alignment of the column heading and the subitem text within the column. It can be one of the following values:

CENTER

The text is centered.

LEFT

The text is left-aligned, which is the default.

RIGHT

The text is right-aligned.

Return value:

The number of the new column, or 0 if this method fails.

Example:

The following example adds three columns to a report list:

```
::method InitReport
  curList = self~GetListControl(102)
  if curList \= .Nil then
  do
    curList~InsertColumn(0,"First Name",50)
    curList~InsertColumn(1,"Last Name",50)
    curList~InsertColumn(2,"Age",50)
  end
```

17.2.18. DeleteColumn

```
>>-aListControl~DeleteColumn(--column--)----------------------><
```

The DeleteColumn method removes a column from a list view control.

Arguments:

The only argument is:

column

The number of the column to be deleted. The first column must be deleted last.

Return value:

0

The column was deleted.

-1

You did not specify *column*.

1

For all other cases.

17.2.19. ModifyColumn

```
>>-aListControl~ModifyColumn(--column--,--+------+--,--width---->
                                          +-text-+

>--+--------------------+--)--------------------------------><
   |       +-LEFT---.    |
   +-,--"--+-CENTER-+--"-+
           +-RIGHT--+
```

The ModifyColumn method sets new attributes for a column of a list view control.

Arguments:

The arguments are:

column

The number of the column. 0 is the first column.

text

The text of the column heading. If you omit this argument, the heading is not changed.

width

The width of the column, in pixels.

align

The alignment of the column heading and the subitem text within the column. It can be one of the following values:

CENTER

The text is centered.

LEFT

The text is left-aligned, which is the default.

RIGHT

The text is right-aligned.

Return value:

0

The column was modified.

-1

You did not specify *column*.

1

For all other cases.

Example:

The following example changes the title, size, and alignment of the first column in a report list:

```
::method ChangeColumn
  curList = self~GetListControl(102)
  curList~ModifyColumn(0,"New Title",100,"RIGHT")
```

17.2.20. ColumnInfo

```
>>-aListControl~ColumnInfo(--column--)------------------------><
```

The ColumnInfo method retrieves the attributes of a column of a list view control.

Arguments:

The only argument is:

column

The number of the column of which the attributes are to be retrieved. 0 is the first column.

Return value:

A compound variable that stores the attributes of the item. If attributes for a column are not available or an error occurs, then the RetStem.!WIDTH variable will be set to 0.

RetStem.!TEXT

The heading of the column.

RetStem.!COLUMN

The column number.

RetStem.!WIDTH

The width of the column.

RetStem.!ALIGN

The alignment of the column: "LEFT", "RIGHT", or "CENTER".

Example:

The following example displays the column attributes in an information box when the column is clicked on:

```
::method OnColumnClick
  use arg id, column
  curList = self~GetListControl(102)
  info. = curList~ColumnInfo(column)
  call InfoDialog("Column Title : " info.!Text"d"x,
                  "Column Number: " info.!Column"d"x,
                  "Column Width : " info.!Width"d"x,
                  "Alignment    : " info.!Align)
```

17.2.21. ColumnWidth

```
>>-aListControl~ColumnWidth(--column--)------------------------><
```

The ColumnWidth method retrieves the width of a column in a report or list view.

Arguments:

The only argument is:

column

The number of the column of which the width is to be retrieved. 0 is the first column.

Return value:

The column width, or -1 if you did not specify *column*, or 0 in all other cases.

Example:

The following example displays the column width in an information box when the column is clicked on:

```
::method OnColumnClick
  use arg id, column
  curList = self~GetListControl(102)
  call InfoDialog(curList~ColumnWidth(column))
```

17.2.22. SetColumnWidth

```
>>-aListControl~SetColumnWidth(--column--+----------+--)------><
                                         +-,--width-+
```

The SetColumnWidth method sets the width of a column in a report or list view.

Arguments:

The arguments are:

column

The number of the column of which the width is to be set. 0 is the first column.

width

The width of the column, in pixels. If you omit this argument, the column is sized automatically.

Return value:

0

The column width was set.

-1

You did not specify *column*.

1

For all other cases.

Example:

The following example enlarges the selected column by 10:

```
::method OnColumnClick
  use arg id, column
  curList = self~GetListControl(102)
  curList~SetColumnWidth(column,curList~ColumnWidth(column)+10)
```

17.2.23. StringWidth

```
>>-aListControl~StringWidth(--text--)-------------------------><
```

The StringWidth method determines the width of a specified string using the current font of the list view control.

Arguments:

The only argument is:

text

The text string of which the width is to be determined.

Return value:

The string width, or -1 if you did not specify a *text*, or 0 in all other cases.

17.2.24. Insert

```
                                    +-0------+              +-0----+
>>-aListControl~Insert(--+------+--,--+--------+--,--text--,--+------+--)-><
                         +-item-+     +-column-+              +-icon-+
```

The Insert method inserts a new item in a list view control.

Arguments:

The arguments are:

item

The number of the item. If you omit this argument, the number of the last item is increased by 1.

column

The number of the column. If you omit this argument, 0 is assumed. This argument only applies to report views.

text

The text of the item.

icon

The index of the icon of the list view item within the bitmap file, set with the SetImages or SetSmallImages method (see page SetSmallImages). The SetImages method must be used for the icon view and the SetSmallImages for the list, report, and small-icon views.

In a report view, this argument can only be used for the first column.

If you omit this argument, 0 is assumed.

Return value:

The index of the new item, or -1 in all other cases.

Example:

The following example inserts items in a list:

```
::method InitReport
  curList = self~GetListControl(102)
  if curList \= .Nil then
  do
    curList~Insert(, ,"First")
    curList~Insert(, ,"Second")
    curList~Insert(, ,"Third")
  end
```

17.2.25. Modify

```
                                      +-0------+
>>-aListControl~Modify(--+------+--,--+--------+--,--text--------->
                         +-item-+     +-column-+

      +-0----+
>--,--+------+--)-----------------------------------------------><
      +-icon-+
```

The Modify method sets some or all attributes of a list view item.

Arguments:

The arguments are:

item

The number of the item. If you omit this argument, the selected item is used.

column

The number of the column. If you omit this argument, 0 is assumed. This argument only applies to report views.

text

The new text for the item.

icon

The new index for the icon of the list view item within the bitmap file, set with the SetImages or SetSmallImages method (see page SetSmallImages). The SetImages method must be used for the icon view and the SetSmallImages method for the list, report, and small-icon views.

In a report view, this argument can only be used for the first column.

If you omit this argument, 0 is assumed.

Return value:

0

The modification was successful.

1

For all other cases.

Example:

The following example modifies the icon of the item that is double-clicked:

```
::method OnActivate
  curList = self~GetListControl(102)
  if curList \= .Nil then
  do
    si = curlist~Focused
    curList~Modify(si, , ,2)
  end
```

17.2.26. SetItemText

```
                                        +-0------+
>>-aListControl~SetItemText(--item--,--+--------+--,--text--)-->&lt;
                                        +-column-+
```

The SetItemText method changes the text of a list view item or a column.

Arguments:

The arguments are:

item

The number of the item.

column

The number of the column. If you omit this argument, 0 is assumed. This argument only applies to report views.

text

The text of the item or a column.

Return value:

0

The change was successful.

-1

You did not specify *column*.

1

For all other cases.

17.2.27. SetItemState

```
>>-aListControl~SetItemState(--item--+-----------------------------+--)-><
                                     |       +-----------------+    |
                                     |       V                 |    |
                                     +-,--"----+-CUT---------+-+--"-+
                                               +-NOTCUT------+
                                               +-DROP--------+
                                               +-NOTDROP-----+
                                               +-FOCUSED-----+
                                               +-NOTFOCUSED--+
                                               +-SELECTED----+
                                               +-NOTSELECTED-+
```

The SetItemState method sets the state of a list view item.

Arguments:

The arguments are:

item

The number of the item.

state

The state of the item, which can be one or more of the following values, separated by blanks:

CUT

The item is marked for a cut-and-paste operation.

NOTCUT

The item cannot be used for a cut-and-paste operation.

DROP

The item is highlighted as a drag-and-drop target.

NOTDROP

The item is not highlighted as a drag-and-drop target.

FOCUSED

The item has the focus and is therefore surrounded by the standard focus rectangle. Only one item can have the focus.

NOTFOCUSED

The item does not have the focus.

SELECTED

The item is selected. Its appearance depends on whether it has the focus and on the system colors used for a selection.

NOTSELECTED

The item is not selected.

Return value:

0

The state was set successfully.

-1

You did not specify *item*.

1

For all other cases.

17.2.28. Add

```
                     +-------+
                     V       |              +-0----+
>>-aListControl~Add(----+---+-+--,--text--,--+------+--)-------><
                       +-,-+                +-icon-+
```

The Add method adds a new item to the report view. It can be used to fill a list view or report view sequentially.

Arguments:

The arguments are:

,

The number of commas specifies in which column the text of the item is to be placed. For example, one comma specifies that the text of the item is to be placed in the first column.

text

The text for the item.

icon

The index of the icon of the list view item within the bitmap file, set with the SetImages or SetSmallImages method (see page SetSmallImages). The SetImages method must be used for the icon view and the SetSmallImages method for the list, report, and small-icon views.

In a report view, this argument can only be used for the first column.

If you omit this argument, 0 is assumed.

Example:

The following example adds three columns and two items to a report list control. To get the following result:

First Name	Last Name	Age
Mike	Miller	30
Sue	Thaxtor	29

you must specify the following:

```
::method InitDialog
  InitDlgRet = self~InitDialog:super

  curList = self~GetListControl("IDC_LIST_REP")
  if curList \= .Nil then
  do
    curList~SetSmallImages("E:\oodlist\oodlist.BMP",16,12)
    curList~InsertColumn(0,"First Name",50)
    curList~InsertColumn(1,"Last Name",50)
    curList~InsertColumn(2,"Age",50)
```

```
    curList~Add("Mike")
    curList~Add(,"Miller")
    curList~Add(, ,"30")
    curList~~Add("Sue")~~Add(,"Thaxton")~~Add(, ,"29")
  end
  return InitDlgRet
```

17.2.29. AddRow

```
                                                +-,----+
                                                V      |
>>-aListControl~AddRow(--+------+--,--icon--,----text-+--)-----><
                         +-item-+
```

The AddRow method adds a new item to a list.

Arguments:

The arguments are:

item

The number of the item. If you omit this argument, the number of the last item is increased by 1.

icon

The index of the icon of the list view item within the bitmap file, set with the SetImages or SetSmallImages method (see page SetSmallImages). The SetImages method must be used for the icon view and the SetSmallImages method for the list, report, and small icon views.

In a report view, this argument can only be used for the first column.

text

Any number of text strings. The first is used for the first column, the second for the second column, and so on. If you specify more text entries than there are columns, the extra entries are ignored.

Return value:

The index of the new item, or -1 in all other cases.

Example:

The following example adds three items to a report list with two columns:

```
::method InitList
  curList = self~GetListControl(101)
  curList~AddRow(, ,"Mike","Miller")
  curList~AddRow(, ,"Sue","Muller")
  curList~AddRow(, ,"Chris","Watson")
```

17.2.30. Delete

```
>>-aListControl~Delete(--item--)------------------------------><
```

The Delete method removes an item from a list view control.

Arguments:

The only argument is:

item

The number of the item.

Return value:

0

The item was deleted.

-1

You did not specify *item*.

1

For all other cases.

Example:

The following example deletes the selected item in a list control:

```
::method DeleteSelectedItem
  curList = self~GetListControl(102)
  curList~Delete(curList~Selected)
```

17.2.31. DeleteAll

```
>>-aListControl~DeleteAll-------------------------------------><
```

The DeleteAll method removes all items from a list view control.

Return value:

0

The items were deleted.

-1

No item was available.

1

For all other cases.

17.2.32. Items

```
>>-aListControl~Items----------------------------------------><
```

The Items method retrieves the number of items in a list view control.

Return value:

The number of items.

17.2.33. Last

```
>>-aListControl~Last-----------------------------------------><
```

The Last method retrieves the number of the last item in a list view control.

Return value:

The number of the last item.

17.2.34. Prepare4nItems

```
>>-aListControl~Prepare4nItems(--items--)---------------------><
```

The Prepare4nItems method prepares a list view control for adding a large number of items.

Arguments:

The only argument is:

items

The number of the items to be added later.

Return value:

0

The list view control was prepared.

-1

You did not specify *items*.

17.2.35. SelectedItems

```
>>-aListControl~SelectedItems--------------------------------><
```

The SelectedItems method determines the number of selected items in a list view control.

Return value:

The number of selected items.

17.2.36. ItemInfo

```
                                   +-0------+
>>-aListControl~ItemInfo(--item--,--+--------+--)--------------><
                                   +-column-+
```

The ItemInfo method retrieves the attributes of a list view item.

Arguments:

The arguments are:

item

The number of the item.

column

The number of the column. If you omit this argument, 0 is assumed.

Return value:

A compound variable that stores the attributes of the item, or -1 in all other cases. The compound variable can be:

RetStem.!TEXT

The item text.

RetStem.!IMAGE

The index of the icon of the list view item within the bitmap file, set with the SetImages or SetSmallImages method (see page SetSmallImages). The SetImages method must be used for the icon view and the SetSmallImages method for the list, report, and small icon views.

In a report view, this argument can only be used for the first column.

RetStem.!STATE

One or more of the following values:

CUT

The item is marked for a cut-and-paste operation.

DROPPED

The item is highlighted as a drag-and-drop target.

FOCUSED

The item has the focus and is therefore surrounded by the standard focus rectangle. Only one item can have the focus.

SELECTED

The item is selected. Its appearance depends on whether it has the focus and on the system colors used for a selection.

Example:

The following example displays the item text, icon index, and the item state in a message box:

```
::method DisplayItemInfo
  curList = self~GetListControl(104)
  itemInfo. = curList~ItemInfo(curList~Selected)
  call InfoDialog("Item Text   : " itemInfo.!Text"d"x,
                  "Image Index : " itemInfo.!Image"d"x,
                  "Item State  : " itemInfo.!State)
```

17.2.37. ItemText

```
                                    +-0------+
>>-aListControl~ItemText(--item--,--+--------+--)--------------><
                                    +-column-+
```

The ItemText method retrieves the text of a list view item or a column.

Arguments:

The arguments are:

item

The number of the item.

column

The number of the column. If you omit this argument, 0 is assumed.

Return value:

The item or column text, or -1 if you did not specify *item*.

17.2.38. ItemState

```
>>-aListControl~ItemState(--item--)---------------------------><
```

The ItemState method retrieves the state of a list view item.

Arguments:

The only argument is:

item

The number of the item.

Return value:

The state of the item, which can be one or more of the following values:

CUT

The item can be used in a cut-and-paste operation.

DROPPED

The item is highlighted as a drag-and-drop target.

FOCUSED

The item has the focus and is therefore surrounded by the standard focus rectangle. Only one item can have the focus.

SELECTED

The item is selected. Its appearance depends on whether it has the focus and on the system colors used for a selection.

17.2.39. Select

```
>>-aListControl~Select(--item--)------------------------------><
```

The Select method selects an item.

Arguments:

The only argument is:

item

The number of the item.

Return value:

0

The item was selected.

-1

You did not specify *item*.

1

For all other cases.

17.2.40. Deselect

```
>>-aListControl~Deselect(--item--)----------------------------><
```

The Deselect method deselects an item.

Arguments:

The only argument is:

item

The number of the item.

Return value:

0

The item was selected.

-1

You did not specify *item*.

1

For all other cases.

17.2.41. Selected

```
>>-aListControl~Selected--------------------------------------><
```

The Selected method returns the number of the selected item.

Return value:

The number of the item selected last, or -1 in all other cases.

17.2.42. LastSelected

```
>>-aListControl~LastSelected----------------------------------><
```

The LastSelected method returns the number of the item selected last.

Return value:

The number of the item selected last, or -1 in all other cases.

17.2.43. Focused

```
>>-aListControl~Focused---------------------------------------><
```

The Focused method retrieves the number of the item that has currently the focus.

Return value:

The number of the item with the focus, or -1 in all other cases.

17.2.44. Focus

```
>>-aListControl~Focus(--item--)------------------------------><
```

The Focus method assigns the focus to the specified item, which is then surrounded by the standard focus rectangle. Although more than one item can be selected, only one item can have the focus.

Arguments:

The only argument is:

item

The number of the item to receive the focus.

Return value:

0

The specified item received the focus.

-1

You did not specify *item*.

1

For all other cases.

17.2.45. DropHighlighted

```
>>-aListControl~DropHighlighted------------------------------><
```

The DropHighlighted method retrieves the item that is highlighted as a drag-and-drop target.

Return value:

The number of the selected item, or -1 in all other cases.

17.2.46. FirstVisible

```
>>-aListControl~FirstVisible----------------------------------><
```

The FirstVisible method retrieves the number of the first item visible in a list or report view.

Return value:

The number of the first item visible, or 0 if the list view control is in icon or small-icon view.

17.2.47. NextSelected

```
>>-aListControl~NextSelected(--item--)------------------------><
```

The NextSelected method retrieves the selected item that follows, or is to the right of, *item*.

Arguments:

The only argument is:

item

The number of the item at which the search is to start. The specified item itself is excluded from the search.

Return value:

The number of the selected item, or -1 in all other cases.

17.2.48. PreviousSelected

```
>>-aListControl~PreviousSelected(--item--)--------------------><
```

The PreviousSelected method retrieves the selected item that precedes, or is to the left of, *item*.

Arguments:

The only argument is:

item

The number of the item at which the search is to start. The specified item itself is excluded from the search.

Return value:

The number of the selected item, or -1 in all other cases.

17.2.49. Next

```
>>-aListControl~Next(--item--)--------------------------------><
```

The Next method retrieves the item that follows, or is to the right of, *item*.

Arguments:

The only argument is:

item

The number of the item at which the search is to start.

Return value:

The number of the following item, or -1 in all other cases.

17.2.50. Previous

```
>>-aListControl~Previous(--item--)----------------------------><
```

The Previous method retrieves the item that precedes, or is to the left of, *item*.

Arguments:

The only argument is:

item

The number of the item at which the search is to start.

Return value:

The number of the previous item, or -1 in all other cases.

17.2.51. NextLeft

```
>>-aListControl~NextLeft(--item--)----------------------------><
```

The NextLeft method retrieves the item left to *item*.

Arguments:

The only argument is:

item

The number of the item at which the search is to start. The specified item itself is excluded from the search.

Return value:

The number of the next item to the left, or -1 in all other cases.

17.2.52. NextRight

```
>>-aListControl~NextRight(--item--)---------------------------><
```

The NextRight method retrieves the item right to *item*.

Arguments:

The only argument is:

item

The number of the item at which the search is to start. The specified item itself is excluded from the search.

Return value:

The number of the next item to the right, or -1 in all other cases.

17.2.53. SmallSpacing

```
>>-aListControl~SmallSpacing----------------------------------><
```

The SmallSpacing method determines the spacing between items in a small-icon list view control.

Return value:

The amount of spacing between the items.

17.2.54. Spacing

```
>>-aListControl~Spacing----------------------------------------><
```

The Spacing method determines the spacing between items in an icon list view control.

Return value:

The amount of spacing between the items.

17.2.55. RedrawItems

```
>>-aListControl~RedrawItems--+-------------------------+------><
                             +-(--first--+---------+--)-+
                                         +-,--last-+
```

The RedrawItems method forces a list view control to redraw a range of items.

Arguments:

The arguments are:

first

The number of the first item to be redrawn. The default is 0.

last

The number of the last item to be redrawn. The default is 0.

Return value:

0

The specified range of items was redrawn.

1

For all other cases.

17.2.56. UpdateItem

```
>>-aListControl~UpdateItem(--item--)--------------------------><
```

The UpdateItem method updates a list view item.

Arguments:

The only argument is:

item

The number of the item to be updated.

Return value:

0

The item was updated.

-1

You did not specify *item*.

1

For all other cases.

17.2.57. Update

```
>>-aListControl~Update----------------------------------------><
```

The Update method updates a list view control.

Return value:

0.

17.2.58. EnsureVisible

```
                                          +-0-+
>>-aListControl~EnsureVisible(--item--,--+---+--)--------------><
                                          +-1-+
```

The EnsureVisible method ensures that a list view item is entirely or partially visible by scrolling the list view control, if necessary.

Arguments:

The arguments are:

item

The number of the item visible.

partial

Specifies whether the item must be entirely visible:

1

The list view control is not scrolled if the item is at least partially visible.

0

The list view control is scrolled if the item is only partially visible. This is the default.

Return value:

0

The item is visible.

-1

You did not specify *item*.

1

For all other cases.

17.2.59. SetSmallImages

```
>>-aListControl~SetSmallImages(--bitmap--+-----------------------------+--)-><
                                         +-,--+-------+--+-----------+-+
                                              +-width-+  +-,--height-+
```

The SetSmallImages method assigns an image list to a small-icon list view control.

Arguments:

The arguments are:

bitmap

The name of, or handle to, a bitmap file that has already been loaded using the LoadBitmap method.

width

The width of each image, in pixels. If you do not specify this argument or specify 0, the width of the image in the image file is used.

height

The height of each image, in pixels. If you do not specify this argument or specify 0, the height of the image in the image file is used.

Return value:

The handle to the image list, or -1 if you did not specify *bitmap*, or 0 in all other cases.

Example:

The following example connects a bitmap file with a small-icon list:

```
::method InitSmallIconList
  curList = self~GetListControl(104)
  curList~SetSmallImages("oodlist.BMP",16,12)
```

17.2.60. SetImages

```
>>-aListControl~SetImages(--bitmap--+-----------------------------+--)-><
                                    +-,--+-------+--+-----------+-+
                                         +-width-+  +-,--height-+
```

The SetImages method assigns an image list to an icon list view control.

Arguments:

The arguments are:

bitmap

The name of, or handle to, a bitmap file that has already been loaded using the LoadBitmap method.

width

The width of each image, in pixels. If you do not specify this argument or specify 0, the width of the image in the image file is used.

height

The height of each image, in pixels. If you do not specify this argument or specify 0, the height of the image in the image file is used.

Return value:

The handle to the image list, or -1 if you did not specify *bitmap*, or 0 in all other cases.

Example:

The following example connects a bitmap file with an icon list:

```
::method InitIconList
  curList = self~GetListControl(104)
```

```
curList~SetImages("oodlist.BMP",16,12)
```

17.2.61. RemoveSmallImages

```
>>-aListControl~RemoveSmallImages----------------------------><
```

The RemoveSmallImages method erases an image list of a small-icon list view control.

Return value:

0

The image list was erased.

1

For all other cases.

17.2.62. RemoveImages

```
>>-aListControl~RemoveImages----------------------------------><
```

The RemoveImages method erases an image list of an icon list view control.

Return value:

0

The image list was erased.

1

For all other cases.

17.2.63. Find

```
                                +- -1--+     +-0-+
>>-aListControl~Find(--text--,--+------+--,--+---+--)----------><
                                +-item-+     +-1-+
```

The Find method searches for a list view item containing *text*. The text of this item must exactly match *text*.

Arguments:

The arguments are:

text

The text of the item to be searched for.

item

Specify the number of the item at which the search is to be started. Specify -1 or omit this argument to start the search at the beginning.

wrap

Specify 1 if the search is to be continued at the beginning if no match is found. Specify 0 or omit this argument if the search is to stop at the end of the list.

Return value:

The number of the item, or -1 in all other cases.

17.2.64. FindPartial

```
                                      +- -1--+     +-0-+
>>-aListControl~FindPartial(--text--,--+------+--,--+---+--)---><
                                      +-item-+     +-1-+
```

The FindPartial method searches for a list view item containing *text*. An item matches if its text begins with *text*.

Arguments:

The arguments are:

text

The text of the item to be searched for.

item

Specify the number of the item at which the search is to be started. Specify -1 or omit this argument to start the search at the beginning.

wrap

Specify 1 if the search is to be continued at the beginning if no match is found. Specify 0 or omit this argument if the search is to stop at the end of the list.

Return value:

The number of the item, or -1 in all other cases.

17.2.65. FindNearestXY

```
>>-aListControl~FindNearestXY(--x--,--y--+--------------------+--)-><
                                          |       +-DOWN--+    |
                                          +-,--"--+-UP----+--"-+
                                                  +-LEFT--+
                                                  +-RIGHT-+
```

The FindNearestXY method searches, in the specified direction, for the item nearest to the specified position.

Arguments:

The arguments are:

x

The x-coordinate of the position at which the search is to be started.

y

The y-coordinate of the position at which the search is to be started.

direction

The direction in which the search should proceed.

Return value:

The index of the item, or -1 in all other cases.

17.2.66. Arrange

```
>>-aListControl~Arrange---------------------------------------><
```

The Arrange method aligns items according to the current alignment style of the list view control.

Return value:

0

The items were aligned.

1

For all other cases.

17.2.67. SnapToGrid

```
>>-aListControl~SnapToGrid------------------------------------><
```

The SnapToGrid method snaps all icons to the nearest grid position.

Return value:

0

The items were snapped.

1

For all other cases.

17.2.68. AlignLeft

```
>>-aListControl~AlignLeft-------------------------------------><
```

The AlignLeft method aligns items along the left window border.

Return value:

0

The items were aligned.

1

For all other cases.

17.2.69. AlignTop

```
>>-aListControl~AlignTop--------------------------------------><
```

The AlignTop method aligns items along the upper window border.

Return value:

0

The items were aligned.

1

For all other cases.

17.2.70. ItemPos

```
>>-aListControl~ItemPos(--item--)----------------------------><
```

The ItemPos method retrieves the position of the upper left corner of the item.

Arguments:

The only argument is:

item

The number of the item.

Return value:

The x- and y-coordinates of the upper left corner of the item, or -1 if you did not specify *item*, or 0 in all other cases.

Note: Use DefListDragHandler to support default dragging:

```
self~ConnectListNotify(104,"BEGINDRAG","DefListDragHandler")
```

17.2.71. SetItemPos

```
                                    +-0-+     +-0-+
>>-aListControl~SetItemPos(--item--,--+---+--,--+---+--)-------><
                                    +-x-+     +-y-+
```

The SetItemPos method moves an item to a specified position in a list view control, which must be in icon or small-icon view.

Arguments:

The arguments are:

item

The number of the item.

x

The x-coordinate of the new position of the upper left corner of the item, in view coordinates. The default is 0.

y

The y-coordinate of the new position of the upper left corner of the item, in view coordinates. The default is 0.

Return value:

0

The item was moved.

-1

You did not specify *item*.

1

For all other cases.

Note: Use DefListDragHandler to support default dragging:

```
self~ConnectListNotify(104,"BEGINDRAG","DefListDragHandler")
```

17.2.72. Edit

```
>>-aListControl~Edit(--item--)--------------------------------><
```

The Edit method begins editing of the text of the specified list view item.

Arguments:

The only argument is:

item

The number of the item.

Return value:

The handle of the edit control used to edit the item text, or 0 in all other cases.

17.2.73. EndEdit

```
>>-aListControl~EndEdit---------------------------------------><
```

The EndEdit method cancels editing of the list view item that is being edited.

17.2.74. SubclassEdit

```
>>-aListControl~SubclassEdit-----------------------------------><
```

The SubclassEdit method is used by the DefListEditHandler to correct an operating system problem if the Esc or Enter key was pressed in an active edit control.

17.2.75. RestoreEditClass

```
>>-aListControl~RestoreEditClass-------------------------------><
```

The RestoreEditClass method is used by the DefListEditHandler to correct an operating system problem if the Esc or Enter key was pressed in an active edit item.

17.2.76. ItemsPerPage

```
>>-aListControl~ItemsPerPage-----------------------------------><
```

The ItemsPerPage method calculates the number of items that vertically fit the visible area of a list view control that is in list or report view. Only fully visible items are counted.

Return value:

The number of fully visible items. If the current view is an icon or small-icon view, the return value is the total number of items in the list view control.

17.2.77. Scroll

```
                        +-0-+     +-0-+
>>-aListControl~Scroll(--+---+--,--+---+--)--------------------><
                        +-x-+     +-y-+
```

The Scroll method scrolls the content of a list view control.

Arguments:

The arguments are:

x

An integer value specifying the amount of horizontal scrolling. If the control is in icon, small-icon, or report view, this value specifies the number of pixels to be scrolled. If it is in list view, this value specifies the number of columns to be scrolled. The default value is 0.

y

An integer value specifying the amount of vertical scrolling. If the control is in icon, small-icon, or report view, this value specifies the number of pixels to be scrolled. If it is in list view, this value specifies the number of lines to be scrolled. The default value is 0.

Return value:

0

Scrolling was successful.

1

Scrolling failed.

17.2.78. BkColor

```
>>-aListControl~BkColor----------------------------------------><
```

The BkColor method retrieves the background color for a list view control.

Return value:

The color-palette index specifier (0 to 18). For more information on color palettes, refer to Definition of Terms.

17.2.79. BkColor=

```
>>-aListControl~BkColor=(--color--)---------------------------><
```

The BkColor= method sets the background color of a list view control.

Arguments:

The only argument is:

color

The new background color. Specify the color-palette index specifier (0 to 18). For more information on color palettes, refer to Definition of Terms.

Example:

The following example sets the background color of a list control to yellow:

```
::method Yellow
  curList = self~GetListControl(104)
  curList~BkColor = 15
  curList~Update
```

17.2.80. TextColor

```
>>-aListControl~TextColor-------------------------------------><
```

The TextColor method retrieves the text color of a list view control.

Return value:

The color-palette index specifier (0 to 18). For more information on color palettes, refer to Definition of Terms.

17.2.81. TextColor=

```
>>-aListControl~TextColor=(--color--)-------------------------><
```

The TextColor= method sets the text color of a list view control.

Arguments:

The only argument is:

color

The new text color. Specify the color-palette index specifier (0 to 18). For more information on color palettes, refer to Definition of Terms.

Example:

The following example sets the text color of a list control to light blue:

```
::method LightBlue
  curList = self~GetListControl(104)
  curList~BkColor = 9
  curList~Update
```

17.2.82. TextBkColor

```
>>-aListControl~TextBkColor------------------------------------><
```

The TextBkColor method retrieves the background color of the text in a list view control.

Return value:

The color-palette index specifier (0 to 18). For more information on color palettes, refer to Definition of Terms.

17.2.83. TextBkColor=

```
>>-aListControl~TextBkColor=(--color--)------------------------><
```

The TextBkColor= method sets the background color for the text in a list view control.

Arguments:

The only argument is:

color

The new background color for text. Specify the color-palette index specifier (0 to 18). For more information on color palettes, refer to Definition of Terms.

17.3. Notification Messages

The list view control sends notification messages to its parent (the dialog) when certain events happen. For more information on notification messages, refer to ConnectListNotify.

The following example shows how to connect the list view notification messages with the corresponding message:

```
::method Init
  use arg InitStem.
  if Arg(1,"o") = 1 then
     InitRet = self~Init:super
  else
     InitRet = self~Init:super(InitStem.)

  if self~Load("list.rc", ) \= 0 then do
     self~InitCode = 1
    return
  end

  /* Connect dialog control items to class methods */
  self~ConnectListNotify("IDC_LIST","Changing","OnChanging_IDC_LIST")
  self~ConnectListNotify("IDC_LIST","Changed","OnChanged_IDC_LIST")
  self~ConnectListNotify("IDC_LIST","DefaultEdit")
  self~ConnectListNotify("IDC_LIST","Delete","OnDelete_IDC_LIST")
  self~ConnectListNotify("IDC_LIST","KeyDown","OnKeyDown_IDC_LIST")
  self~ConnectButton("IDC_PB_NEW","IDC_PB_NEW")
  self~ConnectButton("IDC_PB_DELETE","IDC_PB_DELETE")
  self~ConnectButton(2,"Cancel")
  self~ConnectButton(9,"Help")
  self~ConnectButton(1,"OK")
  return InitRet
```

Chapter 18. ButtonControl Class

The ButtonControl class provides methods to query and modify push button controls. It inherits all methods of the DialogControl class (see DialogControl Class).

Use the GetButtonControl method (see GetButtonControl) to retrieve an object of the ButtonControl class.

Requires:

The ButtonControl class requires the class definition file `oodwin32.cls`:

```
::requires "oodwin32.cls"
```

Methods:

Instances of the ButtonControl class implement the methods listed in the ButtonControl Instance Methods table.

Table 18-1. ButtonControl Instance Methods

Method...	...on page
ChangeBitmap	ChangeBitmap
DimBitmap	DimBitmap
DisplaceBitmap	DisplaceBitmap
DrawBitmap	DrawBitmap
GetBitmapSizeX	GetBitmapSizeX
GetBitmapSizeY	GetBitmapSizeY
GetBmpDisplacement	GetBmpDisplacement
Scroll	Scroll
ScrollBitmapFromTo	ScrollBitmapFromTo
ScrollText	ScrollText
State	State
State=	State=
Style=	Style=

18.1. State

```
>>-aButtonControl~State----------------------------------------><
```

The State method retrieves the current state of the associated button control.

Return value:

A text string that can contain one or more of the following keywords, separated by blanks:

"CHECKED"

The radio button or the check box is checked. A radio button is checked when it contains a black dot, a check box is checked when it contains a check mark.

"UNCHECKED"

The radio button or the check box is not checked. A radio button is not checked when it does not contain a black dot and a check box is not checked when it does not contain a check mark.

"INDETERMINATE"

A 3-state check box button is neither checked nor unchecked. Only 3-state check box buttons can be in this state. When in the indeterminate state, the check box button is drawn in a visually distinctive manner that is different from checked or unchecked. Exactly how the indeterminate state is drawn is dependent on the operating system version.

"PUSHED"

When a button is in the pushed state it is drawn as a sunken button, otherwise it is drawn as a raised button. The user causes a button to be in the pushed state by clicking the button with the left mouse button. As long as the mouse button is held down, the button will be drawn as pushed. When the user releases the mouse button, the button will resume its not pushed state. When this keyword is not in the text string, the button is not in the pushed state.

"FOCUS"

The button has the keyboard focus. When this keyword is missing from the text string, the button does not have the focus.

Example:

```
button = MyDialog~GetButtonControl("IDOK")
if button == .Nil then return
say button~State
```

The result could be "UNCHECKED FOCUS".

18.2. State=

```
                            +-------------------+
                            V                   |
>>-aButtonControl~State=--"----+-CHECKED-------+-+--"---------><
                              +-UNCHECKED-----+
                              +-INDETERMINATE-+
                              +-PUSHED--------+
                              +-NOTPUSHED-----+
```

```
+-FOCUS---------+
```

The State= method sets the state for the associated button control.

Arguments:

The only argument is:

new_State

A text string that contains one or more of the following keywords, separated by a blank:

CHECKED

The radio button or the check box is to be set to the "checked" state.

UNCHECKED

The radio button or the check box is to be set to the "unchecked" state.

INDETERMINATE

The 3-state check box button is to be set to the "indeterminate" state. This state can only be applied to 3-state check box buttons.

PUSHED

The button is to be set to the "pushed" state.

NOTPUSHED

The "pushed" state is to be removed from the button.

FOCUS

The button is to be set to the "focused" state.

Example:

```
button = MyDialog~GetButtonControl("IDOK")
if button == .Nil then return
button~State="FOCUS PUSHED"
```

Notes:

Some points to remember when using this method.

1. A button can be set to only one of the checked, unchecked or indeterminate states. If more than one of these states is specified in the text string the button is set to the state that is first in the string.
2. Only 3-state check box buttons can be set to the indeterminate state. If this keyword is used for a button that is not a 3-state button then the button's state is not changed.

3. To change a button state to not focused programmatically use one of the ooDialog methods that move the focus: TabToNext, TabToPrevious, SetFocus, FocusItem, etc..

18.3. Style=

```
>>-aButtonControl~Style=--"--+-DEFPUSHBUTTON---+--"------------><
                             +-CHECKBOX--------+
                             +-AUTOCHECKBOX----+
                             +-RADIOBUTTON-----+
                             +-AUTORADIOBUTTON-+
                             +-3STATE----------+
                             +-AUTO3STATE------+
                             +-GROUPBOX--------+
                             +-OWNERDRAW-------+
```

The Style= method changes the style of the associated button control.

Arguments:

The only argument is:

new_Style

A text string that contains one of the following keywords:

DEFPUSHBUTTON

A default push button that is pushed when the Enter key is pressed.

CHECKBOX

A check box the state of which has to be maintained by the program.

AUTOCHECKBOX

A check box the check state of which toggles between checked and unchecked each time the user selects the check box.

RADIOBUTTON

A radio button the state of which has to be maintained by the program.

AUTORADIOBUTTON

A radio button that sets the check state of the button to checked and the check state for all other buttons in the same group to unchecked each time the user selects this radio button.

3STATE

A button that is equal to a check box except that the check box can be grayed as well as checked or unchecked.

AUTO3STATE

A button that is equal to a three-state check box except that the check box changes its state when the user selects it. The state cycles through checked, grayed, and unchecked.

GROUPBOX

A rectangle in which other controls can be grouped. A label is displayed in the upper left corner of the rectangle.

OWNERDRAW

An owner-drawn button that can be used to display a bitmap or graphics. If none of the previous values is specified, the push-button style is set for the associated button control.

Example:

The following example makes the OK button the default button:

```
button = MyDialog~GetButtonControl("IDOK")
if button == .Nil then return
button~Style="DEFPUSHBUTTON"
```

18.4. ChangeBitmap

```
>>-aButtonControl~ChangeBitmap(--bmpNormal--,--+------------+--,-->
                                               +-bmpFocused-+

>--+-------------+--,--+-------------+--+-----------------+--)-><
   +-bmpSelected-+     +-bmpDisabled-+  +-,--+-FRAME----+-+
                                            +-USEPAL---+
                                            +-INMEMORY-+
                                            +-STRETCH--+
```

The ChangeBitmap method changes the bitmap of a bitmap button.

Arguments:

The arguments are:

bmpNormal

The (alphanumeric) name, (numeric) resource ID, or handle of a bitmap that is displayed when the button is neither selected, nor focused, nor disabled. If you specify the bitmap handle, the INMEMORY option must be specified.

This option is used if none of the other arguments is specified.

bmpFocused

The (alphanumeric) name, (numeric) resource ID, or handle of a bitmap that is displayed when the button is focused. The focused button is activated when the Enter key is pressed. If you specify the bitmap handle, the INMEMORY option must be specified.

bmpSelected

The (alphanumeric) name, (numeric) resource ID, or handle of a bitmap that is displayed when the button is clicked and held. If you specify the bitmap handle, the INMEMORY option must be specified.

bmpDisabled

The (alphanumeric) name, (numeric) resource ID, or handle of a bitmap that is displayed when the button is disabled. If you specify the bitmap handle, the INMEMORY option must be specified.

styleOptions

The last argument can be one of the following:

FRAME

Draws a frame around the button. When you use this option, the bitmap button behaves like a normal Windows button except that a bitmap is shown instead of text.

USEPAL

Takes the colors of the bitmap file and stores them as the system color palette. This option is needed when the bitmap was created with a palette other than the default Windows color palette. Use it for one button only because only one color palette can be active at a time.

This option is not valid for a bitmap loaded from a dynamic-link library.

INMEMORY

This option must be used if the named bitmaps are already loaded into memory by using the LoadBitmap method (see LoadBitmap). In this case, you must specify a bitmap handle instead of a file name or ID.

STRETCH

If this option is specified and the extent of the bitmap is smaller than the extent of the button rectangle, the bitmap is adapted to match the extent of the button. This option has no effect on bitmaps loaded from a dynamic-link library.

Example:

```
button = MyDialog~GetButtonControl("IDOK")
if button == .Nil then return
button~ChangeBitmap("AddBut_n.bmp", "AddBut_f.bmp", "AddBut_s.bmp", ,
"AddBut_d.bmp", "FRAME")
```

See also ConnectBitmapButton.

18.5. DisplaceBitmap

```
>>-aButtonControl~DisplaceBitmap(--x--,--y--)------------------><
```

The DisplaceBitmap method sets the position of a bitmap within a bitmap button.

Arguments:

The arguments are:

x

The horizontal displacement, in screen pixels. A negative value can also be used.

y

The vertical displacement, in screen pixels. A negative value can also be used.

Example:

The following example moves the bitmap within the associated bitmap button 4 screen pixels to the right and 3 pixels upward:

```
button = MyDialog~GetButtonControl("IDOK")
if button == .Nil then return
parse value button~GetBmpDisplacement with dx dy
button~DisplacementBitmap(244, dx+4, dy-3)
```

18.6. GetBmpDisplacement

```
>>-aButtonControl~GetBmpDisplacement--------------------------><
```

The GetBmpDisplacement method retrieves the position of a bitmap within a bitmap button.

Return value:

The horizontal and vertical positions of the bitmap, in screen pixels and separated by a blank.

Example:

See DisplaceBitmap.

18.7. Scroll

```
>>-aButtonControl~Scroll(--xPos--,--yPos--,--left--,--top--,--right--,--bottom--)-><
```

The Scroll method moves the rectangle within the associated button and redraws the uncovered area with the button background color. It is used to move bitmaps within bitmap buttons.

Arguments:

The arguments are:

xPos, yPos

The new position of the rectangle, in screen pixels.

left, top, right, bottom

The upper left and lower right corner of the rectangle to be moved.

18.8. ScrollText

```
                                                          +-10-------+
>>-aButtonControl~ScrollText(--text--,--+----------+--,--+----------+--,------>
                                        +-fontName-+     +-fontSize-+

                                    +-0---------+     +-4----+     +-10----+
>--+--------------------------+--,--+-----------+--,--+------+--,--+-------+-->
   |    +----------------+    |     +-displaceY-+     +-step-+     +-sleep-+
   |    V                |    |
   +-"----+-THIN-------+-+--"-+
          +-EXTRALIGHT-+
          +-LIGHT------+
          +-MEDIUM-----+
          +-SEMIBOLD---+
          +-EXTRABOLD--+
          +-BOLD-------+
          +-HEAVY------+
          +-UNDERLINE--+
          +-ITALIC-----+
          +-STRIKEOUT--+

      +-0-----+
>--,--+-------+--)-------------------------------------------------------><
      +-color-+
```

The ScrollText method scrolls text in the associated button with the given font, size, and color. The text is scrolled from right to left. If the method is started concurrently, call it a second time to stop scrolling. The associated button must have the OWNERDRAWN style.

Arguments:

The arguments are:

text

A text string that is displayed and scrolled.

fontName

The name of the font used to write the text. If omitted, the system font is used.

fontSize

The size of the font used to write the text. If omitted, the standard size (10) is used.

fontStyle

This argument can be one or more of the keywords listed in the syntax diagram. If you use more than one keyword, put them in one string, separated by blanks.

displaceY

The vertical displacement of the text relative to the top of the client area of the window. The default is 0.

step

The amount of screen pixels that the text is moved in each cycle. The default is 4.

sleep

The time, in milliseconds, that the program waits after each movement to determine the scrolling speed. The default is 10.

color

The color index used for the text. The default is 0, which is black.

Example:

The following example scrolls the string "Hello world!" from left to right within the associated button. The text is located 2 pixels below the top of the client area, one move is 3 screen pixels, and the delay time after each movement is 15 ms.

```
button = MyDialog~GetButtonControl("IFOK")
if button == .Nil then return
button~ScrollText("Hello world!", "Arial", 36, "BOLD ITALIC", 2, 3, 15, 4)
```

18.9. GetBitmapSizeX

```
>>-aButtonControl~GetBitmapSizeX------------------------------><
```

The GetBitmapSizeX method retrieves the width of the bitmap that is set for the associated button.

Return value:

The width of the bitmap, in screen pixels.

18.10. GetBitmapSizeY

```
>>-aButtonControl~GetBitmapSizeY------------------------------><
```

The GetBitmapSizeY method retrieves the height of the bitmap that is set for the associated button.

Return value:

The height of the bitmap, in screen pixels.

18.11. DrawBitmap

```
                              +-0----+     +-0----+     +-0----+
>>-aButtonControl~DrawBitmap(--+------+--,--+------+--,--+------+--,-->
                              +-tarx-+     +-tary-+     +-srcx-+

   +-0----+     +-0-----+     +-0------+
>--+------+--,--+-------+--,--+--------+--)-------------------><
   +-srcy-+     +-width-+     +-height-+
```

The DrawBitmap method draws the bitmap of the associated bitmap button. You can also use this method to move a bitmap or part of it.

Contrary to the method DisplaceBitmap, which sets a permanent position for the bitmap, DrawBitmap immediately draws the bitmap, or part of it, at the specified position. If the button must be refreshed, the bitmap is drawn at the position set with DisplaceBitmap. DrawBitmap is used by ScrollBitmapFromTo, for example.

Arguments:

The arguments are:

tarx,tary

The position relative to the client area of the button where the bitmap is to be displayed. The default is 0,0.

srcx,srcy

The offsets that specify the first pixel in the bitmap to be displayed. If you omit these arguments, the pixel at position 0,0 is the first pixel displayed.

width,height

The width and height of the bitmap. If you omit these arguments or specify 0, the entire bitmap is displayed at the specified position. You can use these arguments to display only parts of the bitmap.

Return value:

0 if the bitmap could be drawn.

Note: You can use the DrawBitmap method to animate a bitmap by providing a bitmap that contains several images and use the offset and extension arguments to display a single image of the bitmap.

18.12. DimBitmap

```
>>-aButtonControl~DimBitmap(--bmpHandle--,--width--,--height--,-->

   +-2-----+     +-2-----+     +-10----+
>--+-------+--,--+-------+--,--+-------+--)-------------------><
   +-stepx-+     +-stepy-+     +-steps-+
```

The DimBitmap method draws a bitmap step by step.

Arguments:

The arguments are:

bmpHandle

A handle to the bitmap loaded with LoadBitmap.

width, height

The extensions of the bitmap.

stepx, stepy

The number of incremental pixels displayed at each step. The default is 2,2.

steps

The number of iterations used to display the bitmap. The default is 10.

Return value:

This method does not return a value.

18.13. ScrollBitmapFromTo

```
>>-aButtonControl~ScrollBitmapFromTo(--fromX--,--fromY--,--toX--,--toY--,-->

>--stepx--,--stepy--,--delay--,--displace--)------------------><
```

The ScrollBitmapFromTo method scrolls the bitmap within the associated bitmap button from one position to another.

For an explanation of the arguments, refer to ScrollBitmapFromTo.

Chapter 19. RadioButton Class

The RadioButton class provides methods to query and modify radio button controls. It inherits all methods of the ButtonControl and DialogControl classes (see ButtonControl Class and DialogControl Class).

Use the GetRadioControl (see GetRadioControl) to retrieve an object of the RadioButton class.

Requires:

The RadioButton class requires the class definition file `oodwin32.cls`:

```
::requires "oodwin32.cls"
```

Methods:

Instances of the RadioControl class implement the methods listed in the RadioButton Instance Methods table.

Table 19-1. RadioButton Instance Methods

Method...	...on page
Check	Check
Indeterminate	Indeterminate
IsChecked	IsChecked
Uncheck	Uncheck

Events for instances of the RadioButton class may be captured using the ConnectButtonNotify method of the MessageExtensions Class.

19.1. IsChecked

```
>>-aRadioButton~IsChecked-------------------------------------><
```

The IsChecked method retrieves the current state of the associated button control.

Return value:

One of the following values:

"CHECKED"

The radio button is selected or the check box is checked.

"UNCHECKED"

The radio button is not selected or the check box is unchecked.

"INDETERMINATE"

The Auto-3-State button is grayed to indicate an indeterminate state.

"UNKNOWN"

No information on the current state available.

19.2. Check

```
>>-aRadioButton~Check----------------------------------------><
```

The Check method marks the associated button control as checked.

19.3. Uncheck

```
>>-aRadioButton~Uncheck---------------------------------------><
```

The Uncheck method deletes the check mark of the button control.

19.4. Indeterminate

```
>>-aRadioButton~Indeterminate---------------------------------><
```

The Indeterminate method grays the check box of an Auto-3-State button to indicate an indeterminate state.

Chapter 20. CheckBox Class

The CheckBox class provides methods to query and modify check box controls. It inherits all methods of:

- The RadioButton class (see RadioButton Class)
- The ButtonControl class (see ButtonControl Class)
- The DialogControl class (see DialogControl Class)

The CheckBox class requires the class definition file `oodwin32.cls`:

```
::requires "oodwin32.cls"
```

Use the GetCheckControl method (see GetCheckControl) to retrieve an object of the CheckBox class.

Events for instances of the CheckBox class may be captured using the ConnectButtonNotify method of the MessageExtensions Class.

Chapter 21. ListBox Class

The ListBox class provides methods to query and modify list box controls. It inherits all methods of the DialogControl class (see page DialogControl Class).

Use the GetListBox method (see page GetListBox) to retrieve an object of the ListBox class.

Requires:

The ListBox class requires the class definition file `oodwin32.cls`:

```
::requires "oodwin32.cls"
```

Methods:

Instances of the ListBox class implement the methods listed in the ListBox Instance Methods table.

Table 21-1. ListBox Instance Methods

Method...	...on page
Add	Add
AddDirectory	AddDirectory
ColumnWidth	ColumnWidth
Delete	Delete
DeleteAll	DeleteAll
DeSelectIndex	DeSelectIndex
DeselectRange	DeselectRange
Find	Find
GetFirstVisible	GetFirstVisible
GetText	GetText
Insert	Insert
ItemHeight	ItemHeight
ItemHeight=	ItemHeight=
Items	Items
MakeFirstVisible	MakeFirstVisible
Modify	Modify
Select	Select
Selected	Selected
SelectedIndex	SelectedIndex
SelectedIndexes	SelectedIndexes
SelectedItems	SelectedItems
SelectIndex	SelectIndex
SelectRange	SelectRange
SetTabulators	SetTabulators
SetWidth	SetWidth

Method...	...on page
Width	Width

21.1. Add

```
>>-aListBox~Add(--listEntry--)--------------------------------><
```

The Add method adds a new item to the list. If the list is not sorted, the new item is added to the end of the list.

Arguments:

The only argument is:

listEntry

A text string added to the list.

Return value:

A one-based index that specifies the position at which the entry has been added, or 0 or a value less than 0 to indicate an error.

21.2. Insert

```
>>-aListBox~Insert(--+-------+--,--listEntry--)----------------><
                     +-index-+
```

The Insert method inserts a new item into the list after the specified item.

Arguments:

The arguments are:

index

The index (starting with 1) of the list item after which the new item is to be added. If this argument is omitted, the list entry is added after the currently selected item.

listEntry

A text string added to the list.

Return value:

A one-based index that specifies the position at which the entry has been added, or 0 or a value less than 0 to indicate an error.

21.3. Delete

```
>>-aListBox~Delete(--+-------+--)-----------------------------><
                     +-index-+
```

The Delete method removes a list item from the associated list box.

Arguments:

The only argument is:

index

The index (starting with 1) of the list item to be removed from the list. If this argument is omitted, the currently selected item is deleted.

Return value:

The number of remaining list items, or 0 to indicate an error.

21.4. DeleteAll

```
>>-aListBox~DeleteAll----------------------------------------><
```

The DeleteAll method removes all list items from the associated list box.

21.5. Find

```
>>-aListBox~Find(--TextorPrefix--+---------------------------------+--)-><
                                 +-,--+------------+--+----------+-+
                                      +-startIndex-+  +-,--exact-+
```

The Find method searches the list box for a list entry containing the specified text or prefix. The search is caseless.

Arguments:

The arguments are:

TextorPrefix

The text or prefix for which the list is searched.

startIndex

The first list item at which the search is to be started. When the search reaches the bottom of the list, it is continued backward. If you omit this argument or specify 0, the entire list is searched.

exact

If you specify 1 or E, the text of the list item must exactly match the text specified for *TextorPrefix*. If you omit this argument or specify 0, the list entries are searched for a prefix that matches *TextorPrefix*.

Return value:

The one-based index of the list entry that matches the search text, or 0 if not found.

21.6. SelectedIndex

```
>>-aListBox~SelectedIndex------------------------------------><
```

The SelectedIndex method retrieves the index of the currently selected list entry. This entry is highlighted and surrounded by a dotted border. To get the selected items for a multiple selection listbox, use GetMultiList; see GetMultiList.

Return value:

The one-based index of the currently selected list entry, or 0 if none is selected.

21.7. Selected

```
>>-aListBox~Selected-----------------------------------------><
```

The Selected method retrieves the text of the currently selected list entry.

Return value:

The text of the currently selected list entry, or an empty string if none is selected.

21.8. SelectIndex

```
>>-aListBox~SelectIndex(--index--)----------------------------><
```

The SelectIndex method selects the list entry at the specified position. The currently selected list item is highlighted and surrounded by a dotted border. To select multiple items of a multiple selection listbox, use SetMultiList; see SetMultiList.

Arguments:

The only argument is:

index

The index (starting with 1) of the list item to be selected. If you specify 0 for this argument, the list box must not contain any selection.

Return value:

0 if an error occurred or you specified 0 for *index* to remove the selection.

21.9. DeSelectIndex

```
>>-aListBox~DeSelectIndex(--+-------+--)----------------------><
                            +-index-+
```

The DeSelectIndex method deselects the list entry at the specified position for multiple selection listboxes. The currently selected list item is highlighted and surrounded by a dotted border.

Arguments:

The only argument is:

index

The index (starting with 1) of the list item to be deselected. If you specify no index or 0 for this argument, all items are deselected.

Return value:

-1 if an error occurred.

21.10. Select

```
>>-aListBox~Select(--ItemText--)------------------------------><
```

The Select method selects the list entry that matches the specified text.

Arguments:

The only argument is:

ItemText

The text that the list box is searched for.

Return value:

0 if an error occurred or a matching list entry was not found.

21.11. SelectRange

```
>>-aListBox~SelectRange(--+-----------------------------+--)---><
                          +-startIndex--+-------------+-+
                                        +-,--endIndex-+
```

The SelectRange method selects one or more consecutive items of the associated multiselection list box.

Arguments:

The arguments are:

startIndex

The position of the first list item to be selected. If this argument is omitted, the selection range starts with the first item in the list.

endIndex

The position of the last list item to be selected. If this argument is omitted, the selection range ends with the last item in the list.

If *startIndex* is equal to *endIndex*, a single item of the list is selected.

Return value:

-1 if an error occurred. This can happen, for example, if the list box is not a multiselection list box.

Example:

The following example selects the items 5 through 9:

```
lb = self~GetListBox("Offers")
if lb == .Nil then return
lb~SelectRange(5,9)
```

21.12. DeselectRange

```
>>-aListBox~DeselectRange(--+-----------------------------+--)-><
                            +-startIndex--+-------------+-+
                                          +-,--endIndex-+
```

The DeselectRange method removes the selection for one or more consecutive items of the associated multiselection list box.

Arguments:

The arguments are:

startIndex

The position of the first list item to be deselected. If this argument is omitted, the deselection range starts with the first item in the list.

endIndex

The position of the last list item to be deselected. If this argument is omitted, the deselection range ends with the last item in the list.

If *startIndex* is equal to *endIndex*, a single item of the list is deselected.

Return value:

-1 if an error occurred. This can happen, for example, if the list box is no multiselection list box.

Example:

The following example deselects the items 1 through 5:

```
lb = self~GetListBox("Offers")
if lb == .Nil then return
lb~DeselectRange(,6)
```

21.13. Items

```
>>-aListBox~Items---------------------------------------------><
```

The Items method retrieves the number of items in the list box.

Return value:

The number of items in the list box.

21.14. SelectedItems

```
>>-aListBox~SelectedItems------------------------------------><
```

The SelectedItems method retrieves the number of items that are currently selected in the associated multiselection list box.

Return value:

The number of selected items, or -1 if this method fails, for example if the list box is no multiselection list box.

Example:

For an example, refer to SelectedIndexes.

21.15. SelectedIndexes

```
>>-aListBox~SelectedIndexes-----------------------------------><
```

The SelectedIndexes method retrieves the indexes of all items that are currently selected in the associated multiselection list box.

Return value:

A text string containing the indexes of the selected items in the list, separated by blanks.

Example:

The following example lists all items selected in the associated multiselection list box:

```
lb = self~GetListBox("Offers")
if lb == .Nil then return
sit = lb~SelectedItems
if sit > 0 then do
   say "You ordered:"
   sndx = lb~SelectedIndexes
   do i = 1 to sit
      parse var sndx order sndx
      say "1 x" lb~GetText(order)
   end
end
```

21.16. MakeFirstVisible

```
>>-aListBox~MakeFirstVisible(--index--)------------------------><
```

The MakeFirstVisible method makes the list entry at the specified position the first visible list item when you scroll up or down.

Arguments:

The only argument is:

index

The one-based index of the list box item to be made first visible.

Return value:

-1 if an error occurred.

21.17. GetFirstVisible

```
>>-aListBox~GetFirstVisible------------------------------------><
```

The GetFirstVisible method retrieves the index of the first list item visible in the list box.

Return value:

The one-based index of the list box item visible first.

21.18. GetText

```
>>-aListBox~GetText(--index--)---------------------------------><
```

The GetText method gets the text of the list item at the specified position in the list box.

Arguments:

The only argument is:

index

The one-based index of the list box item containing the text you are interested in.

Return value:

The text of the list box item at the given position, or an empty string if the *index* does not refer to an item or an error occurred.

21.19. Modify

```
>>-aListBox~Modify(--+-------+--,--newText--)------------------><
                     +-index-+
```

The Modify method changes the text of the list item at the specified position in the list box.

Arguments:

The arguments are:

index

The one-based index of the list box item of which the text is to be changed. If you omit this argument, the currently selected item is modified.

newText

The new text string to be displayed at the given position.

Return value:

The one-based index of the modified list box item at the given position. The return value is 0 if an error occurred, or -1 if the *index* does not refer to an item.

21.20. SetTabulators

```
                           +-,------+
                           V        |
>>-aListBox~SetTabulators(----tabPos-+--)---------------------><
```

The SetTabulators method sets the tabulators for the associated list box control. This enables you to use items containing tab characters ("09"x), which is useful if you want to format the list in more than one column when using proportional fonts.

Arguments:

The only argument is:

tabPos

The position or positions of the tabs relative to the left edge of the list box, in dialog units.

Return value:

1 if an error occurred.

Example:

The following example creates a list that can handle up to three tabulators in a list entry. The tabulator positions are 10, 20, and 30.

```
lb = MyDialog~GetListBox(102)
if lb == .Nil then return
lb~SetTabulators(10, 20, 30)
lb~Add(textcol1 || "09"x || textcol2 || "09"x || textcol3 || "09"x ||,
   textcol4)
```

21.21. AddDirectory

```
                                          +---------------+
                                          V               |
>>-aListBox~AddDirectory(--drvPath--,--"----+-READWRITE-+-+--"--)-><
                                            +-READONLY--+
                                            +-HIDDEN----+
                                            +-SYSTEM----+
                                            +-DIRECTORY-+
                                            +-ARCHIVE---+
```

The AddDirectory method adds all or selected file names of a given directory to the list box.

Arguments:

The arguments are:

drvpath

The drive, path, and name pattern.

fileAttributes

The attributes that the files must have in order to be added:

READWRITE

Normal read/write files (same as none).

READONLY

Files that have the read-only bit.

HIDDEN

Files that have the hidden bit.

SYSTEM

Files that have the system bit.

DIRECTORY

Files that have the directory bit.

ARCHIVE

Files that have the archive bit.

Return value:

The one-based index of the file added last to the list, or 0 if an error occurred.

Example:

The following example puts the names of all read/write files with extension .REX in the given directory of the list box:

```
MyDialog~AddDirectory(203, drive":\"path"\*.rex", "READWRITE")
```

21.22. SetWidth

```
>>-aListBox~SetWidth(--width--)-------------------------------><
```

The SetWidth method sets the internal width of the list box, in dialog units. If the internal width exceeds the width of the list box control and the list box has the HSCROLL style, the list box provides a horizontal scroll bar.

Arguments:

The only argument is:

width

The width of the list box, in dialog units.

Example:

The following example sets the internal width twice the width of the list box control:

```
lb = MyDialog~GetListBox(102)
if lb == .Nil then return
lb~SetWidth(lb~SizeX*2)
```

21.23. Width

```
>>-aListBox~Width----------------------------------------------><
```

The Width method retrieves the internal width of the associated list box.

Return value:

The internal width of the list box, in dialog units, or 0 if an error occurred.

21.24. ItemHeight

```
>>-aListBox~ItemHeight----------------------------------------><
```

The ItemHeight method retrieves the height of the items in the list box, in dialog units.

Return value:

The height of the list box items, in dialog units.

21.25. ItemHeight=

```
>>-aListBox~ItemHeight=--height--------------------------------><
```

The ItemHeight= method sets the height of the items in the list box, in dialog units.

Arguments:

The only argument is:

height

The height of all list box items, in dialog units.

21.26. ColumnWidth=

```
>>-aListBox~ColumnWidth=--width--------------------------------><
```

The ColumnWidth= method sets the width of the list box columns in a multicolumn list box control.

Arguments:

The only argument is:

width

The width of all list box columns, in dialog units.

Chapter 22. ComboBox Class

The ComboBox class provides methods to query and modify combo box controls. It inherits all methods of the DialogControl class (see DialogControl Class).

Use the GetComboBox method (see GetComboBox) to retrieve an object of the ComboBox class.

Requires:

The ComboBox class requires the class definition file `oodwin32.cls`:

```
::requires "oodwin32.cls"
```

Methods:

Instances of the ComboBox class implement the methods listed in the ComboBox Instance Methods table.

Table 22-1. ComboBox Instance Methods

Method...	...on page
Add	Add
AddDirectory	AddDirectory
CloseDropDown	CloseDropDown
Delete	Delete
DeleteAll	DeleteAll
EditSelection	EditSelection
Find	Find
GetText	GetText
Insert	Insert
IsDropDownOpen	IsDropDownOpen
Items	Items
Modify	Modify
OpenDropDown	OpenDropDown
Select	Select
SelectIndex	SelectIndex
Selected	Selected
SelectedIndex	SelectedIndex

22.1. Add

```
>>-aComboBox~Add(--listEntry--)--------------------------------><
```

The Add method adds a new item to the list of the combo box. If the list is not sorted, the new item is added to the end of the list.

Arguments:

The only argument is:

listEntry

A text string added to the list.

Return value:

A one-based index that specifies the position at which the entry has been added, or 0 or a value less than 0 to indicate an error.

22.2. Insert

```
>>-aComboBox~Insert(--+-------+--,--listEntry--)--------------><
                      +-index-+
```

The Insert method inserts a new item into the list of the combo box after the specified item.

Arguments:

The arguments are:

index

The index (starting with 1) of the list item after which the new item is to be added. If this argument is omitted, the list entry is added after the currently selected item.

listEntry

A text string added to the list.

Return value:

A one-based index that specifies the position at which the entry has been added, or 0 or a value less than 0 to indicate an error.

22.3. Delete

```
>>-aComboBox~Delete(--index--)--------------------------------><
```

The Delete method removes a list item from the associated combo box.

Arguments:

The only argument is:

index

The index (starting with 1) of the list item to be removed from the list. If this argument is omitted, the currently selected item is deleted.

Return value:

The number of remaining list items, or 0 to indicate an error.

22.4. DeleteAll

```
>>-aComboBox~DeleteAll----------------------------------------><
```

The DeleteAll method removes all list items from the associated combo box.

Arguments:

The only argument is:

index

The index (starting with 1) of the list item to be removed from the list. If this argument is omitted, the currently selected item is deleted.

Return value:

The number of remaining list items, or 0 to indicate an error.

22.5. Find

```
>>-aComboBox~Find(--TextorPrefix--+---------------------------------+--)-><
                                  +-,--+------------+--+----------+-+
                                       +-startIndex-+  +-,--exact-+
```

The Find method searches the combo box for a list entry containing the specified text or prefix. The search is caseless.

Arguments:

The arguments are:

TextorPrefix

The text or prefix for which the list is searched.

startIndex

The first list item at which the search is to be started. When the search reaches the bottom of the list, it is continued backward. If you omit this argument or specify 0, the entire list is searched.

exact

If you specify 1 or E, the text of the list item must exactly match the text specified for *TextorPrefix*. If you omit this argument or specify 0, the list entries are searched for a prefix that matches *TextorPrefix*.

Return value:

The one-based index of the list entry that matches the search text, or 0 if not found.

22.6. SelectedIndex

```
>>-aComboBox~SelectedIndex------------------------------------><
```

The SelectedIndex method retrieves the index of the currently selected entry of the combo box list. This entry is highlighted and surrounded by a dotted border.

Return value:

The one-based index of the currently selected list entry, or 0 if none is selected.

22.7. Selected

```
>>-aComboBox~Selected-----------------------------------------><
```

The Selected method retrieves the text of the currently selected entry of the combo box list.

Return value:

The text of the currently selected list entry, or an empty string if none is selected.

22.8. SelectIndex

```
>>-aComboBox~SelectIndex(--index--)---------------------------><
```

The SelectIndex method selects the list entry at the specified position. The currently selected list item in the combo box is highlighted and surrounded by a dotted border.

Arguments:

The only argument is:

index

The index (starting with 1) of the list item to be selected. If you specify 0 for this argument, the combo box must not contain any selection.

Return value:

0 if an error occurred or you specified 0 for *index* to remove the selection.

22.9. Select

```
>>-aComboBox~Select(--ItemText--)------------------------------><
```

The Select method selects the list entry that matches the specified text.

Arguments:

The only argument is:

ItemText

The text that the combo box is searched for.

Return value:

0 if an error occurred or a matching list entry was not found.

22.10. Items

```
>>-aComboBox~Items---------------------------------------------><
```

The Items method retrieves the number of items in the combo box.

Return value:

The number of items in the combo box list.

22.11. GetText

```
>>-aComboBox~GetText(--index--)-------------------------------><
```

The GetText method gets the text of the list item at the specified position in the combo box.

Arguments:

The only argument is:

index

The one-based index of the combo box item containing the text you are interested in.

Return value:

The text of the combo box item at the given position, or an empty string if the *index* does not refer to an item or an error occurred.

22.12. Modify

```
>>-aComboBox~Modify(--index--,--newText--)---------------------><
```

The Modify method changes the text of the list item at the specified position in the combo box.

Arguments:

The arguments are:

index

The one-based index of the combo box item of which the text is to be changed. If you omit this argument, the currently selected item is modified.

newText

The new text string to be displayed at the given position.

Return value:

The one-based index of the modified combo box item at the given position. The return value is 0 if an error occurred, or -1 if the *index* does not refer to an item.

22.13. AddDirectory

```
                                          +---------------+
                                          V               |
>>-aComboBox~AddDirectory(--drvPath--,--"----+-READWRITE-+-+--"--)-><
                                            +-READONLY--+
                                            +-HIDDEN----+
                                            +-SYSTEM----+
                                            +-DIRECTORY-+
                                            +-ARCHIVE---+
```

The AddDirectory method adds all or selected file names of a given directory to the combo box.

Arguments:

The arguments are:

drvpath

The drive, path, and name pattern.

fileAttributes

Specify the file attributes the files must possess in order to be added:

READWRITE

Normal read/write files (same as none).

READONLY

Files that have the read-only bit.

HIDDEN

Files that have the hidden bit.

SYSTEM

Files that have the system bit.

DIRECTORY

Files that have the directory bit.

ARCHIVE

Files that have the archive bit.

Return value:

The one-based index of the file name added last to the list, or 0 if an error occurred.

Example:

The following example puts the names of all read/write files with extension .REX in the given directory of the list box:

```
MyDialog~AddDirectory(203, drive":\"path"\*.rex", "READWRITE")
```

22.14. OpenDropDown

```
>>-aComboBox~OpenDropDown------------------------------------><
```

The OpenDropDown method opens the list box of the associated combo box.

22.15. CloseDropDown

```
>>-aComboBox~CloseDropDown------------------------------------><
```

The CloseDropDown method closes the list box of the associated combo box.

22.16. IsDropDownOpen

```
>>-aComboBox~IsDropDownOpen-----------------------------------><
```

The IsDropDownOpen method retrieves whether the list box of the associated combo box is open, that is, visible.

Return value:

1

The list box is open.

0

For all other cases.

22.17. EditSelection

```
>>-aComboBox~EditSelection(--startNdx--,--endNdx--)------------><
```

The EditSelection method selects the specified text range in the edit control of the associated combo box.

Arguments:

The arguments are:

startNdx

The one-based index of the first character in the edit control to be selected. If you omit this argument or specify 0, the selection is removed.

endNdx

The one-based index of the last character in the edit control to be selected. If you omit this argument or specify 0, the selection is removed.

Return value:

0

The selection was successful.

1

An error occurred.

Chapter 23. ScrollBar Class

The ScrollBar class provides methods to query and modify scroll bars. It inherits all methods of the DialogControl class (see page DialogControl Class).

Use the GetScrollBar method to retrieve an object of the ScrollBar class.

Requires:

The ScrollBar class requires the class definition file `oodwin32.cls`:

```
::requires "oodwin32.cls"
```

Methods:

Instances of the ScrollBar class implement the methods listed in the ScrollBar Instance Methods table.

Table 23-1. ScrollBar Instance Methods

Method...	...on page
DeterminePosition	DeterminePosition
Position	Position
Range	Range
SetRange	SetRange
SetPos	SetPos

23.1. SetRange

```
>>-aScrollBar~SetRange(--min--,--max--+-----------+--)--------->&lt;
                                      +-,--redraw-+
```

The SetRange method sets the minimum and maximum positions for the associated scroll bar.

Arguments:

The arguments are:

min

The minimum position to which the scroll bar can be moved.

max

The maximum position to which the scroll bar can be moved.

redraw

If you specify 1 or Y, the scroll bar is redrawn using the new range. Otherwise, the range is set but the scroll bar display is not updated. The default value is 1.

Return value:

0

Setting the range was successful.

1

For all other cases.

23.2. Range

```
>>-aScrollBar~Range-------------------------------------------><
```

The Range method retrieves the minimum and maximum positions of the associated scroll bar.

Return value:

The minimum and maximum positions, separated by a blank.

23.3. SetPos

```
>>-aScrollBar~SetPos(--position--+-----------+--)-------------><
                                +-,--redraw-+
```

The SetPos method sets the position of the scroll box for the associated scroll bar.

Arguments:

The arguments are:

position

The position to which the scroll box is to be moved.

redraw

If you specify 1 or Y, the scroll bar is redrawn using the new position. Otherwise, the new position is set, but the scroll bar display is not updated. The default value is 1.

Return value:

0

Setting the position was successful.

1

For all other cases.

23.4. Position

```
>>-aScrollBar~Position----------------------------------------><
```

The Position method retrieves the position of the scroll box in the associated scroll bar.

Return value:

The position of the scroll box.

23.5. DeterminePosition

```
>>-aScrollBar~DeterminePosition(--posdata----------------------->

>--+-----------------------------------+--)------------------><
   +-,--+------------+--+-------------+-+
        +-singleStep-+  +-,--pageStep-+
```

The DeterminePosition method determines the new position of the scroll box based on the position sent with the scroll bar notification messages.

Arguments:

The arguments are:

posdata

The position sent with the scroll bar notification messages.

singleStep

The value by which the scroll box position is increased or decreased when the user performs a single-step event like using the Down or Up arrow keys or clicking on the arrow buttons of the scroll bar.

pageStep

The value by which the scroll box position is increased or decreased when the user performs a page-step event like using the PgDn or PgUp arrow keys or clicking on an area in the scroll bar that is not occupied by the scroll box or the arrow buttons.

Return value:

The resulting position based on *posdata* and the current position.

Note: The position of a scroll bar cannot be modified directly by a user. The Object Rexx program must react to the notification that is the result of the user interaction and set the resulting position of the scroll box using SetPos. Use the DeterminePosition method to determine the resulting position within your notification handler for the scroll bar notification messages.

Chapter 24. PropertySheet Class

The PropertySheet class provides methods to control a property sheet. It is a subclass of the CategoryDialog class (see CategoryDialog Class). A property sheet is similar to a category dialog that spreads its dialog items over several pages (categories), where the individual pages are controlled by a tab control instead of radio buttons or combo box lists.

Refer to PROPDEMO.REX in the OODIALOG\SAMPLES directory for an example.

The PropertySheet class requires the class definition file `oodwin32.cls`:

```
::requires "oodwin32.cls"
```

Use an object of the DialogControl class or one of its subclasses to work with an individual dialog item of the sheets. To retrieve such an object, you can call one of the following methods depending on the requested control:

- GetStaticControl
- GetEditControl
- GetButtonControl
- GetRadioControl
- GetCheckControl
- GetListBox
- GetComboBox
- GetScrollBar
- GetTreeControl
- GetListControl
- GetProgressBar
- GetSliderControl
- GetTabControl

24.1. Init

```
                                                          +-10---+
>>-aPropertySheet~Init(--+----------+--,--+----------+--,--+------+--,-->
                        +-DlgData.-+     +-cattable-+     +-tabx-+

   +-4----+
>--+------+--,--+-------+--,--+-------+---)---------------------------><
   +-taby-+     +-style-+     +-hFile-+
```

The Init method is called when a new instance of the PropertySheet class is created and initializes the property sheet object.

Arguments:

The arguments are:

DlgData.

An optional stem variable containing the initial values for some or all dialog items. If the dialog is terminated with the OK button, the values in the data fields of the dialog are copied to this variable. The ID of the dialog items is used to name the entry within the stem.

cattable

You can use this argument to set the category names, separated by blanks. Your class, which is a subclass of PropertySheet, must provide a method for each of the categories in the string to define the dialog items of the related category page. The name of the method is equal to the category name in the string.

If you omit this argument, set the category names to the catalog directory in the InitCategories method (see InitCategories).

tabx, taby

The position of the tab control that is used to select the category. The default is 10,4.

style

Determines the style of the property sheet and the tab control used to select the visible category. The style must be specified as a text string that can contain tab control options (see AddTabControl) or the option WIZARD, which adds a backward and a forward button with ID 11 and 12 to the dialog to switch between the category pages.

hFile

The file name of a header file containing symbolic ID defines for resources.

Example:

The following example creates a property sheet dialog with 4 sheets:

```
dlg = MyProperty~new(MyData.,"Movies Cinemas Days Ticket", , ,"WIZARD")
dlg~createCenter(200,180,"Let's go to the movies")
dlg~execute("SHOWTOP")
     .
     .
     .
::class MyProperty subclass PropertySheet
::method Movies    /* define the Movies page */
::method Cinemas   /* define the Cinemas page */
     .
     .
     .
```

An additional example can be found under the `Init` method, in the BaseDialog section. This example shows the use of the dialog data stem and header file arguments in more detail.

Chapter 25. ProgressBar Class

A progress bar is a window that an application can use to indicate the progress of a lengthy operation. It consists of a rectangle that is gradually filled, from left to right, with the system highlight color as an operation progresses. It has a range and a current position. The range represents the entire duration of the operation, and the current position represents the progress that the application has made toward completing the operation.

The ProgressBar class provides methods to change the range and the current position of the progress bar. The window procedure uses the range and the current position to determine the percentage of the progress bar. The highest possible range or current position value is 65,535.

Refer to OODPBAR.REX and PROPDEMO.REX in the OODIALOG\SAMPLES directory for an example.

The ProgressBar class does not send any notification messages. For information about notification messages, refer to ConnectListNotify.

Requires:

The ProgressBar class requires the class definition file `oodwin32.cls`:

```
::requires "oodwin32.cls"
```

Methods:

Instances of the ProgressBar class implement the methods listed in the ProgressBar Instance Methods table.

Table 25-1. ProgressBar Instance Methods

Method...	...on page
SetPos	SetPos
SetRange	SetRange
SetStep	SetStep
Step	Step

25.1. Step

```
>>-aProgressBarControl~Step(--+-----------+--)----------------><
                              +-increment-+
```

The Step method advances the current position of a progress bar or the increment set with the SetStep method (see SetStep), and redraws the bar to reflect the new position.

Arguments:

The only argument is:

increment

The value for advancing the current position. If you do not specify this argument, the position is advanced by the value set with the SetStep method. If the position exceeds the maximum range value, the progress indicator starts at the beginning again.

Return value:

The previous position.

25.2. SetPos

```
>>-aProgressBarControl~SetPos(--newpos--)---------------------><
```

The SetPos method sets the new position for a progress bar and redraws the bar to reflect the new position.

Arguments:

The only argument is:

newpos

The new position.

Return value:

The previous position, or -1 if you do not specify *newpos*.

25.3. SetStep

```
                                +-10------+
>>-aProgressBarControl~SetStep(--+---------+--)----------------><
                                +-newstep-+
```

The SetStep method specifies the step increment for a progress bar. The step increment is the amount by which the progress bar increases its current position whenever the method is called.

Arguments:

The only argument is:

newstep

The new step increment. The default step increment is 10.

Return value:

The previous position, or -1 if you do not specify *newpos*.

25.4. SetRange

```
                                +-0---+     +-100-+
>>-aProgressBarControl~SetRange(--+-----+--,--+-----+--)------><
                                +-min-+     +-max-+
```

The SetRange method sets the minimum and maximum values for a progress bar and redraws the bar to reflect the new range.

Arguments:

The arguments are:

min

The minimum range value. The default is 0.

max

The maximum range value. The default is 100.

Return value:

A string containing the minimum and maximum value of the previous range, separated by a blank, or 0 in all other cases.

Chapter 26. SliderControl Class

A slider control, which is also called a trackbar, is a window that contains a slider and optional tick marks. When the user moves the slider, using either the mouse or the direction keys, the slider sends notification messages to indicate the change.

Sliders are useful if you want the user to select a specific value or a set of consecutive values within a specific range. For example, you might use a slider to enable the user to set the repeat rate of the keyboard by moving the slider to a given tick mark.

The slider in a trackbar moves in increments that you specify when you create it. For example, if you specify that the trackbar should have a range from 0 to 10, the slider can occupy only eleven positions: a position at the left side of the slider and one position for each increment in the range. Typically, each of these positions is identified by a tick mark.

After you have created a slider, you can use the SliderControl methods to set and retrieve its properties. Refer to PROPDEMO.REX in the OODIALOG\SAMPLES directory for an example.

The SliderControl class sends notification messages to notify about the event. For information about notification messages, refer to ConnectListNotify.

Requires:

The SliderControl class requires the class definition file `oodwin32.cls`:

```
::requires "oodwin32.cls"
```

Methods:

Instances of the SliderControl class implement the methods listed in the SliderControl Instance Methods table.

Table 26-1. SliderControl Instance Methods

Method...	...on page
ClearSelRange	ClearSelRange
ClearTicks	ClearTicks
CountTicks	CountTicks
GetLineStep	GetLineStep
GetPageStep	GetPageStep
GetTick	GetTick
InitRange	InitRange
InitSelRange	InitSelRange
Pos	Pos
Pos=	Pos=
Range	Range
SelRange	SelRange
SetLineStep	SetLineStep
SetMax	SetMax

Method...	...on page
SetMin	SetMin
SetPageStep	SetPageStep
SetPos	SetPos
SetSelEnd	SetSelEnd
SetSelStart	SetSelStart
SetTickAt	SetTickAt
SetTickFrequency	SetTickFrequency

26.1. Pos=

```
>>-aSliderControl~Pos=--value---------------------------------><
```

The Pos= method sets the new logical position of the slider and redraws the slider.

Arguments:

The only argument is:

value

The new logical position. A valid position is an integer value within the range of the minimum and maximum positions of the slider. If you specify a value outside this range, the position is set to the maximum or minimum position.

26.2. SetPos

```
                                  +-0-+
>>-aSliderControl~SetPos(--pos--,--+---+--)--------------------><
                                  +-1-+
```

The SetPos method sets the new logical position of the slider and redraws the slider if required.

Arguments:

The arguments are:

pos

The new logical position. A valid position is an integer value within the range of the minimum and maximum positions of the slider. If you specify a value outside this range, the position is set to the maximum or minimum position.

redraw

The redraw flag. If you specify 1, the control is redrawn with the slider at the position given by *pos*. If you specify 0 or omit this argument, the control is not redrawn. However, the new position is set regardless of the redraw argument.

Example:

The following example sets the slider to the maximum position, with the range of the slider already been set to 0 to 100 using the SetRange method:

```
::method SetToMax
  ctrl=self~GetSliderControl("IDC_1")
  ctrl~SetPos(100,1)
```

26.3. Pos

```
>>-aSliderControl~Pos----------------------------------------><
```

The Pos method retrieves the current logical position of the slider.

Return value:

The current logical position of the slider.

Example:

The following example displays the current slider position:

```
::method DisplayPos
  ctrl=self~GetSliderControl("IDC_1")
  pos = ctrl~Pos
  say pos
```

26.4. InitRange

```
                              +-0---+     +-100-+     +-0-+
>>-aSliderControl~InitRange(--+-----+--,--+-----+--,--+---+--)-><
                              +-min-+     +-max-+     +-1-+
```

The InitRange method sets the minimum and maximum positions of the slider and redraws the slider, if required.

Arguments:

The arguments are:

min

The minimum position of the slider. The default is 0.

max

The maximum position of the slider. The default is 100.

redraw

The redraw flag. If you specify 1, the slider is redrawn after the range is set. If you specify 0 or omit this argument, the slider is not redrawn.

Return value:

0

The range was set.

-1

The minimum you specified is greater than the maximum.

Example:

The following example sets the range, the line step, the page step, and the tick frequency of the slider:

```
::method InitDialog
  curSL = self~GetSliderControl("IDC_1")
  if curSL \= .Nil then do
    curSL~InitRange(0,100)
    curSL~SetLineStep(1)
    curSL~SetPageStep(10)
    curSL~SetTickFrequency(10)
  end
```

26.5. SetMin

```
                                  +-1-+
>>-aSliderControl~SetMin(--min--,--+---+--)--------------------><
                                  +-0-+
```

The SetMin method sets the minimum logical position for a slider and redraws the slider, if required.

Arguments:

The arguments are:

min

The minimum position of the slider.

redraw

The redraw flag. If you specify 1 or omit this argument, the slider is redrawn after the minimum position is set. If you specify 0, the slider is not redrawn.

Return value:

0

The minimum position was set.

-1

You omitted *min*.

26.6. SetMax

```
                                  +-1-+
>>-aSliderControl~SetMax(--max--,--+---+--)--------------------><
                                  +-0-+
```

The SetMax method sets the maximum logical position for a slider and redraws the slider, if required.

Arguments:

The arguments are:

min

The maximum position of the slider.

redraw

The redraw flag. If you specify 1 or omit this argument, the slider is redrawn after the maximum position is set. If you specify 0, the slider is not redrawn.

Return value:

0

The maximum position was set successfully.

-1

You omitted *min*.

26.7. Range

```
>>-aSliderControl~Range---------------------------------------><
```

The Range method retrieves the minimum and maximum positions of the slider.

Return value:

The minimum and maximum positions of the slider, separated by a blank.

Example:

The following example displays the range of a slider:

```
::method DisplayRange
  ctrl=self~GetSlidercontrol("IDC_1")
  range = ctrl~Range
  parse var range min max
  say min max
```

26.8. ClearTicks

```
                                +-1-+
>>-aSliderControl~ClearTicks(--+---+--)------------------------><
                                +-0-+
```

The ClearTicks method removes the current tick marks from a slider. It does not, however, remove the first and last tick marks because they are created automatically by the slider.

Arguments:

The arguments are:

redraw

The redraw flag. If you specify 1 or omit this argument, the slider is redrawn after the tick marks are removed. If you specify 0, the slider is not redrawn.

Return value:

0.

26.9. CountTicks

```
>>-aSliderControl~CountTicks----------------------------------><
```

The CountTicks method retrieves the number of tick marks in a slider, including the first and last tick marks, which are created automatically by the slider.

Return value:

The number of tick marks.

26.10. GetTick

```
>>-aSliderControl~GetTick(--tic--)-----------------------------><
```

The GetTick method retrieves the logical position of a tick mark in a slider. A valid position is an integer value within the minimum and maximum positions of the slider.

Arguments:

The only argument is:

tic

A zero-based index identifying a tick mark. A valid index is in the range of 0 to 2 ticks less than the tick count returned by the CountTicks method (see CountTicks).

Return value:

The logical position of the specified tick mark, or -1 if you did not specify a valid index for *tic*.

26.11. SetTickAt

```
>>-aSliderControl~SetTickAt(--pos--)--------------------------><
```

The SetTickAt method sets a tick mark at the specified logical position in the slider.

Arguments:

The only argument is:

pos

An integer value within the minimum and maximum positions of the slider.

Return value:

0

The tick mark was set.

-1

You omitted *pos*.

1

For all other cases.

26.12. SetTickFrequency

```
                                    +-1----+
>>-aSliderControl~SetTickFrequency(--+------+--)---------------><
                                    +-freq-+
```

The SetTickFrequency method sets the interval frequency for tick marks in a slider. For example, if you set the frequency to 2, a tick mark is displayed for every other increment in the slider's range.

Arguments:

The only argument is:

freq

The frequency of the tick marks. The default is 1, that is, every increment in the range is associated with a tick mark.

Return value:

0

The tick mark frequency was set.

-1

You omitted *freq*.

1

For all other cases.

Example:

The following example sets the range, the line step, the page step, and the tick frequency of the slider:

```
::method InitDialog
  curSL = self~GetSliderControl("IDC_1")
  if curSL \= .Nil then do
    curSL~InitRange(0,100)
    curSL~SetLineStep(1)
    curSL~SetPageStep(10)
    curSL~SetTickFrequency(10)
  end
```

26.13. GetLineStep

```
>>-aSliderControl~GetLineStep---------------------------------><
```

The GetLineStep method retrieves the number of logical positions that the slider moves in response to keyboard input from the arrow keys, such as the Right arrow or the Down arrow keys. The logical positions are the integer increments within the minimum and maximum positions of the slider.

Return value:

The line size for the slider.

26.14. GetPageStep

```
>>-aSliderControl~GetPageStep---------------------------------><
```

The GetPageStep method retrieves the number of logical positions that the slider moves in response to keyboard input, such as the PageUp or PageDown keys, or mouse input, such as clicks in the slider's

channel. The logical positions are the integer increments within the minimum and maximum positions of the slider.

Return value:

The page size for the slider.

26.15. SetLineStep

```
>>-aSliderControl~SetLineStep(--step--)------------------------><
```

The SetLineStep method sets the number of logical positions that the slider moves in response to keyboard input from the arrow keys, such as the Right arrow or the Down arrow keys. The logical positions are the integer increments within the minimum and maximum positions of the slider.

Arguments:

The only argument is:

step

The new line size.

Return value:

The previous line size, or -1 if you omit *step*.

Example:

The following example sets the range, the line step, the page step, and the tick frequency of the slider:

```
::method InitDialog
  curSL = self~GetSliderControl("IDC_1")
  if curSL \= .Nil then do
    curSL~InitRange(0,100)
    curSL~SetLineStep(1)
    curSL~SetPageStep(10)
    curSL~SetTickFrequency(10)
  end
```

26.16. SetPageStep

```
>>-aSliderControl~SetPageStep(--step--)------------------------><
```

The SetPageStep method sets the number of logical positions that the slider moves in response to keyboard input, such as the PageUp or PageDown keys, or mouse input, such as clicks in the slider's

channel. The logical positions are the integer increments within the minimum and maximum positions of the slider.

Arguments:

The only argument is:

step

The new page size.

Return value:

The previous page size.

Example:

The following example sets the range, the line step, the page step, and the tick frequency of the slider:

```
::method InitDialog
  curSL = self~GetSliderControl("IDC_1")
  if curSL \= .Nil then do
    curSL~InitRange(0,100)
    curSL~SetLineStep(1)
    curSL~SetPageStep(10)
    curSL~SetTickFrequency(10)
  end
```

26.17. InitSelRange

```
                                +-0---+                 +-0------+
>>-aSliderControl~InitSelRange(--+-----+--,--+-----+--,--+--------+--)-><
                                +-min-+     +-max-+     +-redraw-+
```

The InitSelRange method sets the starting and ending logical positions for the current selection range in a slider. It is ignored if the slider does not have a selection range.

Arguments:

The arguments are:

min

The logical starting position of the selection range. The default is 0.

max

The logical ending position of the selection range. If you omit this argument, the maximum position of the slider's range is assumed.

redraw

The redraw flag. If you specify 1, the slider is redrawn after the selection range is set. If you specify 0 or omit this argument, the selection range is set but the slider is not redrawn.

Return value:

-1

The minimum you specified is greater than the maximum.

0

In all other cases.

26.18. SetSelStart

```
                                        +-1------+
>>-aSliderControl~SetSelStart(--min--,--+--------+--)---------><
                                        +-redraw-+
```

The SetSelStart method sets the starting logical position for the current selection range in a slider. It is ignored if the slider does not have a selection range.

Arguments:

The arguments are:

min

The logical starting position of the selection range.

redraw

The redraw flag. If you specify 1 or omit this argument, the slider is redrawn after the starting position has been set. If you specify 0, the starting position is set but the slider is not redrawn.

Return value:

-1

You omitted *min*.

0

In all other cases.

26.19. SetSelEnd

```
                                    +-1------+
>>-aSliderControl~SetSelEnd(--max--,--+--------+--)------------><
                                    +-redraw-+
```

The SetSelEnd method sets the logical ending position for the current selection range in a slider. It is ignored if the slider does not have a selection range.

Arguments:

The arguments are:

min

The logical ending position of the selection range.

redraw

The redraw flag. If you specify 1 or omit this argument, the slider is redrawn after the ending position has been set. If you specify 0, the ending position is set but the slider is not redrawn.

Return value:

-1

You omitted *max*.

0

In all other cases.

26.20. ClearSelRange

```
>>-aSliderControl~ClearSelRange(--redraw--)-------------------><
```

The ClearSelRange method clears the current selection range in a slider.

Arguments:

The only argument is:

redraw

The redraw flag. If you specify 1, the slider is redrawn after the selection is cleared.

Return value:

0.

26.21. SelRange

```
>>-aSliderControl~SelRange------------------------------------><
```

The SelRange method retrieves the starting position of the current selection range in a slider.

Return value:

The starting and ending positions of the current selection range, separated by a blank.

Chapter 27. TabControl Class

A tab control can be compared to a divider in a notebook or a label in a file cabinet. By using a tab control, an application can define several pages for the same area of a dialog or dialog control. Each page consists of a set of information or a group of controls that the application displays when the user selects the corresponding tab.

A special type of tab control displays tabs that look like buttons. Clicking a button immediately performs a command instead of displaying a page.

You can apply specific characteristics to tab controls by specifying tab control styles. For example, you can specify the alignment and general appearance of the tabs in a tab control.

By default, a tab control displays only one row of tabs. If not all tabs can be shown at once, the tab control displays an up-and-down control so that the user can scroll to view additional tabs.

Refer to PROPDEMO.REX in the OODIALOG\SAMPLES directory for an example.

The TabControl class sends notification messages to notify about the event. For information about notification messages, refer to ConnectListNotify.

Requires:

The TabControl class requires the class definition file `oodwin32.cls`:

```
::requires "oodwin32.cls"
```

Methods:

Instances of the TabControl class implement the methods listed in the TabControl Instance Methods table.

Table 27-1. TabControl Instance Methods

Method...	...on page
AddFullSeq	AddFullSeq
AddSequence	AddSequence
AdjustToRectangle	AdjustToRectangle
Delete	Delete
DeleteAll	DeleteAll
Focus	Focus
Focused	Focused
Insert	Insert
ItemInfo	ItemInfo
Items	Items
Last	Last
Modify	Modify
PosRectangle	PosRectangle
RemoveImages	RemoveImages

Method...	...on page
RequiredWindowSize	RequiredWindowSize
Rows	Rows
Select	Select
SelectIndex	SelectIndex
Selected	Selected
SelectedIndex	SelectedIndex
SetImages	SetImages
SetPadding	SetPadding
SetSize	SetSize

27.1. Insert

```
>>-aTabControl~Insert(--+-----+--,--text--,--icon--,--numValue--)-><
                        +-tab-+
```

The Insert method inserts a new tab in a tab control.

Arguments:

The arguments are:

tab

The number of the tab. If you omit this argument, the number of the last tab is increased by 1, starting with 0.

text

The label text for the inserted tab.

icon

The index of the icon in the image list of the tab control, set with the SetImages method (see SetImages).

numValue

An integer value stored together with the tab to save information.

Return value:

The number of the new tab, or -1 for all other cases.

Example:

The following example inserts three tabs in a tab control with the specified text and a specific item:

```
::method InitDialog
  InitDlgRet = self~InitDialog:super
  curTab = self~GetTabControl("ID_TAB")
  if curTab \= .Nil then do
    curTab~SetImages("oodtab.BMP",16,16)
    curTab~Insert(,"First Tab",0)
    curTab~Insert(,"Second Tab",1)
    curTab~Insert(,"Third Tab",2)
  end
return InitDlgRet
```

27.2. Modify

```
>>-aTabControl~Modify(--tab--,--text--,--icon--,--numValue--)-->&lt;
```

The Modify method sets some or all of the attributes of a tab.

Arguments:

The arguments are:

tab

The number of the tab.

text

The label text for the tab.

icon

The index of the icon in the image list of the tab control, set with the SetImages method (see SetImages).

numValue

An integer value stored together with the tab to save information.

Return value:

0

The attributes were set.

-1

You did not specify *tab*.

1

In all other cases.

27.3. AddSequence

```
>>-aTabControl~AddSequence(--text1--,--text2--,--text3--,--...--)-><
```

The AddSequence method inserts a sequence of tabs in a tab control for which you can only specify the label text. The number of the tab inserted last is increased by 1.

Arguments:

The only argument is:

text

The label text for the inserted tab.

Return value:

The number of the tab inserted last, or -1 for all other cases.

Example:

The following example inserts three tabs in a tab control:

```
::method InitDialog
  InitDlgRet = self~InitDialog:super
  curTab = self~GetTabControl("ID_TAB")
  if curTab \= .Nil then do
    curTab~AddSequence("First Tab","Second Tab","Third Tab")
  end
return InitDlgRet
```

27.4. AddFullSeq

```
>>-aTabControl~AddFullSeq(--text_1--,--icon_1--,--numValue_1--,--text_2-->

>--,--icon_2--,--numValue_2--,--...--)---------------------------------><
```

The AddFullSeq method inserts a sequence of tabs in a tab control for which you can specify the number, label text, and integer value.

Arguments:

The arguments are:

text

The label text for the inserted tab.

icon

The index of the icon in the image list of the tab control, set with the SetImages method (see SetImages).

numValue

An integer value stored together with the tab to save information.

Return value:

The number of the tab inserted last, or -1 for all other cases.

Example:

The following example adds a sequence of tabs and sets their text and icon:

```
::method InitDialog
  InitDlgRet = self~InitDialog:super
  curTab = self~GetTabControl("ID_TAB")
  if curTab \= .Nil then do
    curTab~SetImages("oodtab.BMP",16,16)
    curTab~AddFullSeq("s11", 0, , "s12", 1, , "s13", 2, , "s14", 3)
  end
return InitDlgRet
```

27.5. Items

```
>>-aTabControl~Items------------------------------------------><
```

The Items method retrieves the number of tabs in a tab control.

Return value:

The number of the tabs, or 0 for all other cases.

Example:

The following example displays the number of tabs:

```
::method DisplayTabNum
  curTab = self~GetTabControl("ID_TAB")
  if curTab \= .Nil then do
    say curTab~Items
  end
```

27.6. Rows

```
>>-aTabControl~Rows------------------------------------------><
```

The Rows method retrieves the current number of rows of tabs in a tab control. Only tab controls with multiline style can have several rows of tabs.

Return value:

The number of the tab rows.

27.7. ItemInfo

```
>>-aTabControl~ItemInfo(--tab--)------------------------------><
```

The ItemInfo method retrieves information about a tab in a tab control.

Arguments:

The only argument is:

text

The number of the tab.

Return value:

A compound variable that stores the attributes of the tab, or -1 in all other cases. The compound variable can be:

RetStem.!TEXT

The label text for the tab.

RetStem.!IMAGE

The index of the tab in the image list of the tab control, or -1 if the tab does not have an image.

RetStem.!PARAM

An integer value stored together with the tab to save information:

Example:

The following example displays the text of all tabs:

```
::method DisplayText
  curTab = self~GetTabControl("ID_TAB")
  if curTab \= .Nil then do
    do i = 0 to curTab~Items - 1
       ItemInfo. = curTab~ItemInfo(i)
```

```
      say ItemInfo.!Text
    end
  end
```

27.8. Delete

```
>>-aTabControl~Delete(--tab--)--------------------------------><
```

The Delete method removes a tab from a tab control.

Arguments:

The only argument is:

tab

The number of the tab to be removed.

Return value:

0

The tab was removed.

-1

You did not specify *tab* or there is no tab available.

1

For all other cases.

27.9. DeleteAll

```
>>-aTabControl~DeleteAll--------------------------------------><
```

The DeleteAll method removes all tabs from a tab control.

Return value:

0

The tabs were removed.

1

For all other cases.

27.10. Last

```
>>-aTabControl~Last-------------------------------------------><
```

The Last method retrieves the number of the last tab in a tab control.

Return value:

The number of the last tab, or 0 in all other cases.

27.11. Selected

```
>>-aTabControl~Selected---------------------------------------><
```

The Selected method retrieves the label text of the currently selected tab.

Return value:

The label text of the currently selected tab, or 0 in all other cases.

27.12. SelectedIndex

```
>>-aTabControl~SelectedIndex----------------------------------><
```

The SelectedIndex method retrieves the number of the currently selected tab.

Return value:

The number of the currently selected tab, or 0 in all other cases.

27.13. Select

```
>>-aTabControl~Select(--text--)-------------------------------><
```

The Select method selects the tab with the specified label text.

Arguments:

The only argument is:

text

The label text of the tab to be selected.

Return value:

The number of the selected tab, or 0 in all other cases.

27.14. SelectIndex

```
>>-aTabControl~SelectIndex(--tab--)---------------------------><
```

The SelectIndex method selects the specified tab in a tab control.

Arguments:

The only argument is:

tab

The number of the tab to be selected.

Return value:

The number of the previously selected tab, or -1 in all other cases.

27.15. Focus

```
>>-aTabControl~Focus(--tab--)---------------------------------><
```

The Focus method sets the focus to the specified tab in a tab control.

Arguments:

The only argument is:

tab

The number of the tab to receive the focus.

Return value:

0.

27.16. Focused

```
>>-aTabControl~Focused----------------------------------------><
```

The Focused method returns the number of the tab that has the focus. The tab with the focus can differ from the selected tab.

Return value:

The number of the tab having the focus.

27.17. SetImages

```
                                    +-0-----+     +-0------+
>>-aTabControl~SetImages(--bitmap--,--+-------+--,--+--------+--)-><
                                    +-width-+     +-height-+
```

The SetImages method assigns an image list to a tab control.

Arguments:

The arguments are:

bitmap

The name of, or the handle to, a bitmap file that is already loaded with LoadBitmap method (see LoadBitmap).

width

The width, in pixels, of each image. If you specify 0 or omit this argument, the width of the image in the image file is used.

height

The height, in pixels, of each image. If you specify 0 or omit this argument, the height of the image in the image file is used.

Return value:

The handle to the previous image list, or 0 for all other cases.

27.18. RemoveImages

```
>>-aTabControl~RemoveImages----------------------------------><
```

The RemoveImages method deletes an image list of a tab control.

Return value:

0

The image list was removed.

1

In all other cases.

27.19. SetPadding

```
>>-aTabControl~SetPadding(--padX--,--padY--)-------------------><
```

The SetPadding method sets the amount of space (padding) around the icon and the label of a tab.

Arguments:

The arguments are:

padX

The amount of horizontal padding, in pixels.

padY

The amount of vertical padding, in pixels.

Return value:

0.

27.20. SetSize

```
>>-aTabControl~SetSize(--width--,--height--)-------------------><
```

The SetSize method sets the width and height of tabs in a fixed-width or owner-drawn tab control.

Arguments:

The arguments are:

width

The new width, in pixels.

height

The new height, in pixels.

Return value:

The old width and height as a string, in pixels.

27.21. PosRectangle

```
>>-aTabControl~PosRectangle(--tab--)--------------------------><
```

The PosRectangle method retrieves the rectangle around a tab in a tab control.

Arguments:

The only argument is:

tab

The number of the tab.

Return value:

A string containing the coordinates of the rectangle, or an empty string. The coordinates are separated by blanks and are in the following order:

- X-coordinate of the upper left corner of the rectangle
- Y-coordinate of the upper left corner of the rectangle
- X-coordinate of the lower right corner of the rectangle
- Y-coordinate of the lower right corner of the rectangle

27.22. AdjustToRectangle

```
>>-aTabControl~AdjustToRectangle(--left--,--top--,--right--,--bottom--)-><
```

The AdjustToRectangle method calculates the window rectangle of a tab control that corresponds to the specified display rectangle.

Arguments:

The arguments are:

left

The x-coordinate of the upper left corner of the display rectangle.

top

The y-coordinate of the upper left corner of the display rectangle.

right

The x-coordinate of the lower right corner of the display rectangle.

bottom

The y-coordinate of the lower right corner of the display rectangle.

Return value:

A string containing the coordinates of the window rectangle, or an empty string. The coordinates are separated by blanks and are in the following order:

- X-coordinate of the upper left corner of the rectangle
- Y-coordinate of the upper left corner of the rectangle
- X-coordinate of the lower right corner of the rectangle
- Y-coordinate of the lower right corner of the rectangle

27.23. RequiredWindowSize

```
>>-aTabControl~RequiredWindowSize(--left--,--top--,--right--,--bottom--)-><
```

The RequiredWindowSize method calculates the display rectangle of a tab control that corresponds to the specified window rectangle.

Arguments:

The arguments are:

left

The x-coordinate of the upper left corner of the window rectangle.

top

The y-coordinate of the upper left corner of the window rectangle.

right

The x-coordinate of the lower right corner of the window rectangle.

bottom

The y-coordinate of the lower right corner of the window rectangle.

Return value:

A string containing the coordinates of the display rectangle, or an empty string. The coordinates are separated by blanks and are in the following order:

- X-coordinate of the upper left corner of the rectangle
- Y-coordinate of the upper left corner of the rectangle
- X-coordinate of the lower right corner of the rectangle
- Y-coordinate of the lower right corner of the rectangle

Chapter 28. TreeControl Class

The tree view control is a dialog that displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files and directories on a disk. Each item consists of a label and an optional image bitmap, and can have a list of subitems associated with it. By clicking on an item, the user can expand and collapse the associated list of subitems.

Refer to OODTREE.REX in the OODIALOG\SAMPLES directory for an example.

Requires:

The TreeControl class requires the class definition file `oodwin32.cls`:

```
::requires "oodwin32.cls"
```

Methods:

Instances of the TreeControl class implement the methods listed in the TreeControl Instance Methods table.

Table 28-1. TreeControl Instance Methods

Method...	...on page
Add	Add
Child	Child
Collapse	Collapse
CollapseAndReset	CollapseAndReset
Delete	Delete
DeleteAll	DeleteAll
DropHighlight	DropHighlight
DropHighlighted	DropHighlighted
Edit	Edit
EndEdit	EndEdit
EnsureVisible	EnsureVisible
Expand	Expand
FirstVisible	FirstVisible
HitTest	HitTest
Indent	Indent
Indent=	Indent=
Insert	Insert
IsAncestor	IsAncestor
ItemInfo	ItemInfo
Items	Items
MakeFirstVisible	MakeFirstVisible
Modify	Modify

Method...	...on page
MoveItem	MoveItem
Next	Next
NextVisible	NextVisible
Parent	Parent
Previous	Previous
PreviousVisible	PreviousVisible
RemoveImages	RemoveImages
RestoreEditClass	RestoreEditClass
Root	Root
Select	Select
Selected	Selected
SetImages	SetImages
SortChildren	SortChildren
SubclassEdit	SubclassEdit
Toggle	Toggle
VisibleItems	VisibleItems

28.1. Methods of the TreeControl Class

The following sections describe the individual methods of the TreeControl class.

28.1.1. Insert

```
                                  +-"LAST"--+     +-""---+     +-0-----+
>>-aTreeControl~Insert(--parent--,--+-after---+--,--+------+--,--+-------+--,-->
                                  +-"FIRST"-+     +-text-+     +-image-+
                                  +-"SORT"--+

                                                     +-"0"-+
>--+----------+--,--+-----------------------+--,--+-----+--)--><
   +-selImage-+     |    +--------------+    |     +-"1"-+
                    |    V              |    |
                    +-"----+-EXPANDED-+-+--"-+
                           +-BOLD-----+
```

The Insert method inserts a new item in a tree view control.

Arguments:

The arguments are:

parent

The handle to the parent item. If you specify "ROOT", the item is inserted at the root of the tree view control.

after

The handle to the item after which the new item is to be inserted or one of the following values:

"FIRST"

Inserts the item at the beginning of the list.

"LAST"

Inserts the item at the end of the list. This is the default.

"SORT"

Inserts the item into the list in alphabetical order.

text

The text for the item. If you omit this argument, "" is assumed.

image

The index of the icon image in the tree view control's bitmap file to be used when the item is in the non-selected state. If you omit this argument, the icon with index 0 is used.

selImage

The index of the icon image in the tree view control's bitmap file to be used when the item is in the selected state. If you omit this argument, the icon specified for *image* is used.

state

Specifies the appearance and functionality of the item. It can be a combination of the following values, separated by a blank:

EXPANDED

The item is currently expanded with all child items visible. This only applies to parent items.

BOLD

The item is shown in bold.

If you omit this argument, the item is inserted in collapsed and normal form.

children

Indicates whether the item has child items associated with it:

"0"

The item has no child items, which is the default.

"1"

The item has one or more child items.

You can use this argument to show an item that does not have any child items, with an expand button. This allows you to save memory usage by dynamically loading and displaying the child items only when the user expands the item.

Return value:

The handle to the new item, or 0 for all other cases.

Example:

The following example inserts child items when a specific root item is expanded. The text strings for the items are loaded from a file.

```
::method OnExpanding_IDC_TREE
  use arg tree, item
  itemFile = "root6.inp"
  curTree = self~GetTreeControl("IDC_TREE")
  itemInfo. = curTree~ItemInfo(item)

  if itemInfo.!TEXT = "Root 6" & \itemInfo.!STATE~POS("EXPANDED") then
  do
    do while lines(itemFile)
      line = linein(itemFile)
      command = "curTree~Insert(item, ,"||line||")"
      interpret command
    end
    curTree~Expand(item)
  end
```

28.1.2. Add

```
                      +-------+
                      V       |     +-""---+     +-0-----+
>>-aTreeControl~Add(----+---+-+--,--+------+--,--+-------+--,--->
                        +-,-+       +-text-+     +-image-+

                                                   +-"0"-+
>--+----------+--,--+-------------------------+--,--+-----+--)-->&lt;
   +-selImage-+     |    +--------------+    |     +-"1"-+
                    |    V              |    |
                    +-"----+-EXPANDED-+-+--"-+
                           +-BOLD-----+
```

The Add method adds a new item to a tree view control.

Arguments:

The arguments are:

,

The number of commas specifies at which parent the item is to be inserted. If you omit this argument, the item is inserted as a root item. Each additional comma inserts the item one level deeper than the item inserted previously. See the example in the following.

text

The text for the item. If you omit this argument, "" is assumed.

image

The index of the icon image in the tree view control's bitmap file to be used when the item is in the non-selected state. If you omit this argument, the icon with index 0 is used.

selImage

The index of the icon image in the tree view control's bitmap file to be used when the item is in the selected state. If you omit this argument, the icon specified for *image* is used.

state

Specifies the appearance and functionality of the item. It can be a combination of the following values, separated by a blank:

EXPANDED

The item is currently expanded with all child items visible. This only applies to parent items.

BOLD

The item is shown in bold.

If you omit this argument, the item is inserted in collapsed and normal form.

children

Indicates whether the item has child items associated with it:

"0"

The item has no child items, which is the default.

"1"

The item has one or more child items.

You can use this argument to show an item that does not have any child items, with an expand button. This allows you to save memory usage by dynamically loading and displaying the child items only when the user expands the item.

Return value:

The handle to the new item, or 0 for all other cases.

Example:

To get the following tree view:

- Peter
  - Mike
    - George
    - Monique
      - John
    - Chris
  - Maud
  - Ringo
- Paul
  - Dave
  - Sam
  - Jeff
- Mary
  - Helen
  - Michelle
  - Diana

your example must look as follows:

```
::method InitDialog
  InitDlgRet = self~InitDialog:super
  curTree = self~GetTreeControl("IDC_TREE")
  if curTree \= .Nil then
  do
    curTree~Add("Peter", , ,"BOLD EXPANDED")
    curTree~Add(,"Mike", , ,"EXPANDED")
    curTree~Add(, ,"George")
    curTree~Add(, ,"Monique")
```

```
        curTree~Add(, , ,"John")
        curTree~Add(, ,"Chris")
        curTree~Add(,"Maud")
        curTree~Add(,"Ringo")
        curTree~Add("Paul", , ,"BOLD EXPANDED")
        curTree~Add(,"Dave")
        curTree~Add(,"Sam")
        curTree~Add(,"Jeff")
        curTree~Add("Mary", , ,"BOLD EXPANDED")
        curTree~Add(,"Helen")
        curTree~Add(,"Michelle")
        curTree~Add(,"Diana")
    end
```

28.1.3. Modify

```
                                  +-""---+     +-0-----+
>>-aTreeControl~Modify(--hItem--,--+------+--,--+-------+--,--+----------+--,-->
                                  +-text-+     +-image-+     +-selImage-+

                                        +-"0"-+
>--+-------------------------------+--,--+-----+--)------------><
   |    +---------------------+    |     +-"1"-+
   |    V                     |    |
   +-"----+-BOLD------------+-+--"-+
          +-NOTBOLD---------+
          +-DROP------------+
          +-NOTDROP---------+
          +-SELECTED--------+
          +-NOTSELECTED-----+
          +-CUT-------------+
          +-NOTCUT----------+
          +-EXPANDEDONCE----+
          +-NOTEXPANDEDONCE-+
          +-EXPANDED--------+
          +-NOTEXPANDED-----+
```

The Modify method sets some or all attributes of an item of a tree view control.

Arguments:

The arguments are:

hItem

The handle to the item to be modified.

text

The text for the item. If you omit this argument, "" is assumed.

image

The index of the icon image in the tree view control's bitmap file to be used when the item is in the non-selected state. If you omit this argument, the icon with index 0 is used.

selImage

The index of the icon image in the tree view control's bitmap file to be used when the item is in the selected state. If you omit this argument, the icon specified for *image* is used.

state

Specifies the appearance and functionality of the item. It can be one or more of the following values, separated by blanks:

BOLD

The item is shown in bold.

NOTBOLD

The item is not bold.

DROP

The item is selected as a drag-and-drop target.

NOTDROP

The item is not selected as a drag-and-drop target.

SELECTED

The item is selected. Its appearance depends on whether it has the focus and whether the system colors are used for the selection.

NOTSELECTED

The item is not selected.

CUT

The item is selected as part of a cut-and-paste operation.

NOTCUT

The item is not selected as part of a cut-and-paste operation.

EXPANDEDONCE

The item's list of child items has been expanded at least once.

NOTEXPANDEDONCE

The item's list of child items has not been expanded at least once.

EXPANDED

The item's list is currently expanded with all child items visible. This only applies to parent items.

NOTEXPANDED

The item's list is currently not expanded.

children

Indicates whether the item has child items associated with it:

"0"

The item has no child items, which is the default.

"1"

The item has one or more child items.

You can use this argument to show an item that does not have any child items, with an expand button. This allows you to save memory usage by dynamically loading and displaying the child items only when the user expands the item.

Return value:

0

The item has been modified.

-1

For all other cases.

Example:

The following example changes the text of the item to bold when it is selected:

```
::method OnSelChanging_IDC_TREE
  curTree = self~GetTreeControl("IDC_TREE")
  curTree~Modify(curTree~selected, , , ,"BOLD")
```

28.1.4. ItemInfo

```
>>-aTreeControl~ItemInfo(--hItem--)---------------------------><
```

The ItemInfo method retrieves some or all attributes of an item of a tree view control.

Arguments:

The only argument is:

hItem

The handle to the item of which attributes are to be retrieved.

Return value:

A compound variable that stores the attributes of the item, or -1 in all other cases. The compound variable can be:

RetStem.!TEXT

The text of the item.

RetStem.!CHILDREN

1

The item has children.

0

The item has no children.

RetStem.!IMAGE

The index of the icon image in the tree view control's bitmap file used when the item is in the non-selected state.

RetStem.!SELECTEDIMAGE

The index of the icon image in the tree view control's bitmap file used when the item is in the selected state.

RetStem.!STATE

An empty string or one or more of the following strings, separated by blanks:

EXPANDED

The item's list is currently expanded with all child items visible. This only applies to parent items.

BOLD

The item is in bold.

SELECTED

The item is selected.

EXPANDEDONCE

The item's list has been expanded at least once. This only applies to parent items.

INDROP

The item is selected as a drag-and-drop target.

Example:

The following example displays the attributes of the selected item:

```
::method Info
  curTree = self~GetTreeControl("IDC_TREE")
  itemInfo. = curTree~ItemInfo(curTree~Selected)
  say itemInfo.!TEXT
  say itemInfo.!CHILDREN
  say itemInfo.!IMAGE
  say itemInfo.!STATE
```

28.1.5. Items

```
>>-aTreeControl~Items-----------------------------------------><
```

The Items method retrieves the number of items in a tree view control.

Return value:

The number of items.

Example:

The following example counts all items in a tree view control:

```
::method Count
  curTree = self~GetTreeControl("IDC_TREE")
  say curTree~Items
```

28.1.6. VisibleItems

```
>>-aTreeControl~VisibleItems----------------------------------><
```

The VisibleItems method obtains the number of items that can be fully visible in a tree view control. This number can be greater than the number of items in the control. The control calculates this value by dividing the height of the client window by the height of an item.

Return value:

The number of items that can be fully visible. For example, if you can see all of 19 items and part of another item, the return value is 19, not 20.

Example:

The following example returns the number of items that can be fully visible:

```
::method Visible
  curTree = self~GetTreeControl("IDC_TREE")
  say curTree~VisibleItems
```

28.1.7. Root

```
>>-aTreeControl~Root------------------------------------------><
```

The Root method retrieves the first or topmost item of the tree view control.

Return value:

The handle to the first item, or 0 in all other cases.

Example:

The following example displays the name of the root item:

```
::method RootName
  curTree = self~GetTreeControl("IDC_TREE")
  itemInfo. = curTree~ItemInfo(curTree~Root)
  say ItemInfo.!Text
```

28.1.8. Parent

```
>>-aTreeControl~Parent(--hItem--)-----------------------------><
```

The Parent method retrieves the parent of the specified item.

Arguments:

The only argument is:

hItem

The handle to the item for which the parent is to be retrieved.

Return value:

The handle to the parent item, or -1 if *hItem* is not specified or is 0, or 0 in all other cases.

Example:

The following example displays the name of the selected item's parent:

```
::method Parent
  curTree = self~GetTreeControl("IDC_TREE")
  itemInfo. = curTree~ItemInfo(curTree~Parent(curTree~Selected))
  say ItemInfo.!Text
```

28.1.9. Child

```
>>-aTreeControl~Child(--hItem--)------------------------------><
```

The Child method retrieves the first child item of *hItem*.

Arguments:

The only argument is:

hItem

The handle to the item of which the first child is to be retrieved.

Return value:

The handle to the first child item, or -1 if you omitted *hItem*, or 0 in all other cases.

Example:

The following example displays the name of parent of the selected item:

```
::method Child
  curTree = self~GetTreeControl("IDC_TREE")
  itemInfo. = curTree~ItemInfo(curTree~Child(curTree~Selected))
  say ItemInfo.!Text
```

28.1.10. Selected

```
>>-aTreeControl~Selected--------------------------------------><
```

The Selected method retrieves the currently selected item.

Return value:

The handle to the currently selected item, or 0 in all other cases.

Example:

The following example displays the name of the selected item:

```
::method SelectedName
```

```
  curTree = self~GetTreeControl("IDC_TREE")
  itemInfo. = curTree~ItemInfo(curTree~Selected)
  say ItemInfo.!Text
```

28.1.11. DropHighlighted

```
>>-aTreeControl~DropHighlighted--------------------------------><
```

The DropHighlighted method retrieves the item that is the target of a drag-and-drop operation.

Return value:

The handle to the item, or 0 in all other cases.

28.1.12. FirstVisible

```
>>-aTreeControl~FirstVisible-----------------------------------><
```

The FirstVisible method retrieves the first visible item in the client window of a tree view control.

Return value:

The handle to the first visible item, or 0 in all other cases.

Example:

The following example displays the name of the first visible item:

```
::method FirstVisibleName
  curTree = self~GetTreeControl("IDC_TREE")
  itemInfo. = curTree~ItemInfo(curTree~FirstVisible)
  say ItemInfo.!Text
```

28.1.13. Next

```
>>-aTreeControl~Next(--hItem--)--------------------------------><
```

The Next method retrieves the sibling item next to *hItem*.

Arguments:

The only argument is:

hItem

The handle to the item next to the sibling item to be retrieved.

Return value:

The handle to the next sibling item, or -1 if you omitted *hItem*, or 0 in all other cases.

Example:

The following example displays the name of the selected item and its siblings:

```
::method SiblingNames
  curTree = self~GetTreeControl("IDC_TREE")
  nextItem = curTree~Selected
  do while nextItem \= 0
    itemInfo. = curTree~ItemInfo(nextItem)
    say ItemInfo.!Text
    nextItem = curTree~Next(nextItem)
  end
```

28.1.14. NextVisible

```
>>-aTreeControl~NextVisible(--hItem--)------------------------><
```

The NextVisible method retrieves the visible item following *hItem*.

Arguments:

The only argument is:

hItem

The handle to the item that precedes the visible item to be retrieved. *hItem* must also be visible.

Return value:

The handle to the next visible item, or -1 if you omitted *hItem*, or 0 in all other cases.

28.1.15. Previous

```
>>-aTreeControl~Previous(--hItem--)---------------------------><
```

The Previous method retrieves the sibling item preceding *hItem*.

Arguments:

The only argument is:

hItem

The handle to the item that follows the sibling item to be retrieved.

Return value:

The handle to the previous sibling item, or -1 if *hItem* is not specified or is 0, or 0 in all other cases.

28.1.16. PreviousVisible

```
>>-aTreeControl~PreviousVisible(--hItem--)--------------------><
```

The PreviousVisible method retrieves the visible item preceding *hItem*.

Arguments:

The only argument is:

hItem

The handle to the item that follows the visible child item to be retrieved. *hItem* must also be visible.

Return value:

The handle to the previous visible child item, or -1 if *hItem* is not specified or is 0, or 0 in all other cases.

28.1.17. Delete

```
>>-aTreeControl~Delete(--hItem--)-----------------------------><
```

The Delete method removes an item from a tree view control.

Arguments:

The only argument is:

hItem

The handle to the item to be deleted.

Return value:

0

The item was deleted.

1

An error occurred.

-1

hItem is 0 or is not a valid value.

Example:

The following example deletes the selected item and all its children, if any:

```
::method IDC_PB_DELETE
  curTree = self~GetTreeControl("IDC_TREE")
  curTree~Delete(curTree~Selected)
```

28.1.18. DeleteAll

```
>>-aTreeControl~DeleteAll-------------------------------------><
```

The DeleteAll method removes all items from a tree view control.

Return value:

0

The items were removed.

1

For all other cases.

28.1.19. Collapse

```
>>-aTreeControl~Collapse(--hItem--)----------------------------><
```

The Collapse method collapses the list of child items associated with the specified parent item.

Arguments:

The only argument is:

hItem

The handle to the parent item to collapse.

Return value:

0

The list of child items has collapsed.

-1

hItem is not specified or is 0.

1

For all other cases.

Example:

The following example collapses the selected item:

```
::method CollapseSelected
  curTree = self~GetTreeControl("IDC_TREE")
  curTree~Collapse(curTree~Selected)
```

28.1.20. CollapseAndReset

```
>>-aTreeControl~CollapseAndReset(--hItem--)-------------------><
```

The CollapseAndReset method collapses the list of child items associated with the specified parent item and removes the child items.

Arguments:

The only argument is:

hItem

The handle to the parent item to collapse.

Return value:

0

The list of child items has collapsed and the child items have been removed.

-1

hItem is not specified or is 0.

1

For all other cases.

Example:

The following example collapses the selected item and removes all its child items:

```
::method CollapseSelectedAndReset
  curTree = self~GetTreeControl("IDC_TREE")
  curTree~CollapseAndReset(curTree~Selected)
```

28.1.21. Expand

```
>>-aTreeControl~Expand(--hItem--)----------------------------><
```

The Expand method expands the list of child items associated with the specified parent item.

Arguments:

The only argument is:

hItem

The handle to the parent item to be expanded.

Return value:

0

The parent item was expanded.

-1

hItem is not specified or is 0.

1

For all other cases.

Example:

The following example expands the selected item:

```
::method ExpandSelected
  curTree = self~GetTreeControl("IDC_TREE")
  curTree~Expand(curTree~Selected)
```

28.1.22. Toggle

```
>>-aTreeControl~Toggle(--hItem--)----------------------------><
```

The Toggle method collapses the list of the specified item if it was expanded, or expands it if it was collapsed.

Arguments:

The only argument is:

hItem

The handle to the item to be expanded or collapsed.

Return value:

0

The item was expanded or collapsed.

-1

hItem is not specified or is 0.

1

For all other cases.

Example:

The following example toggles between expanding and collapsing a selected item:

```
::method ToggleSelected
  curTree = self~GetTreeControl("IDC_TREE")
  curTree~Toggle(curTree~Selected)
```

28.1.23. EnsureVisible

```
>>-aTreeControl~EnsureVisible(--hItem--)----------------------><
```

The EnsureVisible method ensures that a tree view item is visible, expanding the parent item or scrolling the tree view control, if necessary.

Arguments:

The only argument is:

hItem

The handle to the item to be visible.

Return value:

0

The items in the tree view control were scrolled to ensure that the specified item is visible.

-1

hItem is not specified or is 0.

1

For all other cases.

28.1.24. Indent

```
>>-aTreeControl~Indent----------------------------------------><
```

The Indent method retrieves the amount, in pixels, by which the child items are indented relative to their parent item.

Return value:

The amount indented, in pixels.

28.1.25. Indent=

```
>>-aTreeControl~Indent=--indent--------------------------------><
```

The Indent= method sets the width of indentation for a tree view control and redraws the control to reflect the new width.

Arguments:

The only argument is:

indent

The width of the indentation, in pixels. If you specify a width that is smaller than the system-defined minimum, it is set to the system-defined minimum.

Return value:

-1 if *indent* is 0.

28.1.26. Edit

```
>>-aTreeControl~Edit(--hItem--)------------------------------><
```

The Edit method starts editing the text of the specified item by replacing the text with a single-line edit control containing this text. It implicitly selects and focuses the specified item.

Arguments:

The only argument is:

hItem

The handle to the item to be edited.

Return value:

The handle to the edit control used to edit the item text, or -1 if *hItem* is not specified or is 0, or 0 in all other cases.

28.1.27. EndEdit

```
>>-aTreeControl~EndEdit(--+--------+--)------------------------><
                          +-cancel-+
```

The EndEdit method ends the editing of the item label of the tree view.

Arguments:

The only argument is:

cancel

Indicates whether editing is canceled without being saved to the label. If you specify "1" or "YES", editing is canceled. Otherwise, the changes are saved to the label, which is the default.

Return value:

0

Editing has ended successfully.

1

For all other cases.

28.1.28. SubclassEdit

```
>>-aTreeControl~SubclassEdit---------------------------------><
```

The SubclassEdit method is used by the DefTreeEditHandler to correct the problem occurring when Esc or the Enter key is pressed in an active edit item.

28.1.29. RestoreEditClass

```
>>-aTreeControl~RestoreEditClass------------------------------><
```

The RestoreEditClass method is used by the DefTreeEditHandler to correct the problem occurring when Esc or the Enter key is pressed in an active edit item.

28.1.30. Select

```
>>-aTreeControl~Select(--hItem--)-----------------------------><
```

The Select method selects a specific item.

Arguments:

The only argument is:

hItem

The handle to the item to be selected.

Return value:

0

The item was selected.

-1

hItem was not specified or is 0.

1

For all other cases.

28.1.31. MakeFirstVisible

```
>>-aTreeControl~MakeFirstVisible(--hItem--)-------------------><
```

The MakeFirstVisible method ensures that *hItem* is visible and displays it at the top of the control's dialog, if possible. If the specified item is near the end of the control's hierarchy of items, it might not become the first visible item depending on how many items fit in the dialog.

Arguments:

The only argument is:

hItem

The handle to the item to be visible first.

Return value:

0

The item is visible first.

-1

hItem was not specified or is 0.

1

For all other cases.

28.1.32. DropHighlight

```
>>-aTreeControl~DropHighlight(--hItem--)----------------------><
```

The DropHighlight method redraws *hItem* in the style used to indicate the target of a drag-and-drop operation.

Arguments:

The only argument is:

hItem

The handle of the item to be redrawn.

Return value:

0

The item was redrawn.

-1

hItem was not specified or is 0.

1

For all other cases.

28.1.33. SortChildren

```
>>-aTreeControl~SortChildren(--hItem--)------------------------><
```

The SortChildren method sorts the child items of the specified parent item in a tree view control.

Arguments:

The only argument is:

hItem

The handle to the parent item the child items of which are to be sorted.

Return value:

0

The child items were sorted.

-1

hItem was not specified or is 0.

1

For all other cases.

28.1.34. SetImages

```
                                    +-0-----+     +-0------+
>>-aTreeControl~SetImages(--hItem--,--+-------+--,--+--------+--)-><
                                    +-width-+     +-height-+
```

The SetImages method sets the image list for a tree view control and redraws the control using the new images.

Arguments:

The arguments are:

bitmap

The name of, or handle to, a bitmap file that is already loaded using the LoadBitmap method (see LoadBitmap).

width

The width of each image, in pixels. If you specify 0 or omit this argument, the height of the image in the image file is used as width.

height

The height of each image, in pixels. If you specify 0 or omit this argument, the height of the image in the image file is used.

Return value:

The handle to the previous image list, or -1 if you did not specify *bitmap*, or 0 in all other cases.

Example:

The following example sets the image list during dialog initialization:

```
::method InitDialog
```

```
expose bmpFile
InitDlgRet = self~InitDialog:super
curTree = self~GetTreeControl("IDC_TREE")
if curTree \= .Nil then
do
  curTree~SetImages(bmpFile,16,12)
end
return InitDlgRet
```

28.1.35. RemoveImages

```
>>-aTreeControl~RemoveImages----------------------------------><
```

The RemoveImages method destroys the image list of the tree view.

Return value:

0

The image list was destroyed.

1

For all other cases.

28.1.36. HitTest

```
>>-aTreeControl~HitTest(--x--,--y--)---------------------------><
```

The HitTest method determines the location of the specified point relative to the client area of a tree view control.

Arguments:

The arguments are:

x

The x-coordinate of the point.

y

The y-coordinate of the point.

Return value:

0 if no item occupies the point, or one or more of the following strings if an item occupies the specified point:

handle

The handle to the item that occupies the specified point.

ABOVE

Above the client area.

BELOW

Below the client area.

NOWHERE

In the client area but below the last item.

ONITEM

On the bitmap or label associated with an item.

ONBUTTON

On the button associated with an item.

ONICON

On the icon associated with an item.

ONINDENT

In the indentation associated with an item.

ONLABEL

On the label (string) associated with an item.

ONRIGHT

In the area to the right of an item.

ONSTATEICON

On the state icon for a tree view item that is in a user-defined state.

TOLEFT

To the left of the client area.

TORIGHT

To the right of the client area.

28.1.37. MoveItem

```
>>-aTreeControl~MoveItem(--hItem--,--hNewParent--,--redraw--+---------------+--)-><
                                                            +-,--"NODELETE"-+
                                                            +-,--"SIBLINGS"-+
```

The MoveItem method moves an item to a new location.

Arguments:

The arguments are:

hItem

The handle to the item to be moved.

hNewParent

The handle to the new parent to which the item is to be moved.

redraw

The tree view control is updated.

options

One of the following options:

"NODELETE"

The item is copied to another location.

"SIBLINGS"

Siblings are moved together with the item.

Return value:

The handle to the new parent, or 0 in all other cases.

28.1.38. IsAncestor

```
>>-aTreeControl~IsAncestor(--hParent--,--hItem--)-------------><
```

The IsAncestor method checks if an item is an ancestor of another item.

Arguments:

The arguments are:

hParent

The ancestor.

hItem

The item to be checked.

Return value:

1 if *hParent* is an ancestor of *hItem*.

28.2. Notification Messages

The tree view control sends notification messages to notify about events. For more information on notification messages, refer to ConnectListNotify.

The following example shows how to connect the tree view notification messages with the corresponding message:

```
::method Init
  use arg InitStem.
  if Arg(1,"o") = 1 then
     InitRet = self~Init:super
  else
     InitRet = self~Init:super(InitStem.)

  if self~Load("tree.rc", ) \= 0 then do
     self~InitCode = 1
    return
  end

  /* Connect dialog control items to class methods */
  self~ConnectTreeNotify("IDC_TREE","SelChanging","OnSelChanging_IDC_TREE")
  self~ConnectTreeNotify("IDC_TREE","SelChanged","OnSelChanged_IDC_TREE")
  self~ConnectTreeNotify("IDC_TREE","Expanding","OnExpanding_IDC_TREE")
  self~ConnectTreeNotify("IDC_TREE","Expanded","OnExpanded_IDC_TREE")
  self~ConnectTreeNotify("IDC_TREE","DefaultEdit")
  self~ConnectTreeNotify("IDC_TREE","Delete","OnDelete_IDC_TREE")
  self~ConnectTreeNotify("IDC_TREE","KeyDown","OnKeyDown_IDC_TREE")
  self~ConnectButton("IDC_PB_NEW","IDC_PB_NEW")
  self~ConnectButton("IDC_PB_DELETE","IDC_PB_DELETE")
  self~ConnectButton(2,"Cancel")
  self~ConnectButton(9,"Help")
  self~ConnectButton(1,"OK")
  return InitRet
```

Chapter 29. VirtualKeyCodes Class

The methods of the VirtualKeyCodes class can be used for all objects of the WindowsProgramManager and WindowObject classes. These classes inherit from the VirtualKeyCodes class. The VirtualKeyCodes class cannot be used as a standalone class.

The VirtualKeyCodes class requires the class definition file `WinSystm.cls`:

```
::requires "WinSystm.cls"
```

29.1. Methods of the VirtualKeyCodes Class

Instances of the VirtualKeyCodes class implement the methods described in the following sections.

29.1.1. VCode

```
>>-aVirtualKeyCodes~VCode(--keyname--)------------------------><
```

The VCode method returns the decimal value of a symbolic key name.

Arguments:

The only argument is:

keyname

The symbolic key name. See Symbolic Names for Virtual Keys for a list of key names.

Return value:

The decimal value of the symbolic key name. If the symbolic name is not found, 255 is returned.

29.1.2. KeyName

```
>>-aVirtualKeyCodes~KeyName(--vcode--)------------------------><
```

The KeyName method returns the symbolic key name of the specified hexadecimal code.

Arguments:

The only argument is:

vcode

The hexadecimal code of the key.

Return value:

The symbolic key name of the specified code.

Example:

The following example deletes or inserts an item in a tree view depending on the selected key:

```
::method OnKeyDown_IDC_TREE
  use arg treeId, key
  curTree = self~GetTreeControl(treeId)
  /* if the Delete key is pressed, delete the selected item */
  if self~KeyName(key) = "DELETE" then
     curTree~Delete(curTree~Selected)
  else
  /* if the Insert key is pressed, simulate pressing the New button */
  if self~KeyName(key) = "INSERT" then
     self~IDC_PB_NEW
```

29.2. Symbolic Names for Virtual Keys

Symbolic Names for Virtual Keys shows the symbolic names and the keyboard equivalents for the virtual keys used by Object Rexx:

Table 29-1. Symbolic Names for Virtual Keys

Symbolic Name	Mouse or Keyboard Equivalent
LBUTTON	Left mouse button
RBUTTON	Right mouse button
CANCEL	Control-break processing
MBUTTON	Middle mouse button (three-button mouse)
BACK	BACKSPACE key
TAB	TAB key
CLEAR	CLEAR key
RETURN	ENTER key
SHIFT	SHIFT key
CONTROL	CRTL key
MENU	ALT key
PAUSE	PAUSE key
CAPITAL	CAPS LOCK key
ESCAPE	ESC key
SPACE	SPACEBAR
PRIOR	PAGE UP key
NEXT	PAGE DOWN key
END	END key

Symbolic Name	Mouse or Keyboard Equivalent
HOME	HOME key
LEFT	LEFT ARROW key
UP	UP ARROW key
RIGHT	RIGHT ARROW key
DOWN	DOWN ARROW key
SELECT	SELECT key
EXECUTE	EXECUTE key
SNAPSHOT	PRINT SCREEN key
INSERT	INS key
DELETE	DEL key
HELP	HELP key
0	0 key
1	1 key
2	2 key
3	3 key
4	4 key
5	5 key
6	6 key
7	7 key
8	8 key
9	9 key
A	A key
B	B key
C	C key
D	D key
E	E key
F	F key
G	G key
H	H key
I	I key
J	J key
K	K key
L	L key
M	M key
N	N key
O	O key
Q	Q key
R	R key

Symbolic Name	Mouse or Keyboard Equivalent
S	S key
T	T key
U	U key
V	V key
W	W key
X	X key
Y	Y key
Z	Z key
NUMPAD0	Numeric keypad 0 key
NUMPAD1	Numeric keypad 1 key
NUMPAD2	Numeric keypad 2 key
NUMPAD3	Numeric keypad 3 key
NUMPAD4	Numeric keypad 4 key
NUMPAD5	Numeric keypad 5 key
NUMPAD6	Numeric keypad 6 key
NUMPAD7	Numeric keypad 7 key
NUMPAD8	Numeric keypad 8 key
NUMPAD9	Numeric keypad 9 key
MULTIPLY	Multiply key
ADD	Add key
SEPARATOR	Separator key
SUBTRACT	Subtract key
DECIMAL	Decimal key
DIVIDE	Divide key
F1	F1 key
F2	F2 key
F3	F3 key
F4	F4 key
F5	F5 key
F6	F6 key
F7	F7 key
F8	F8 key
F9	F9 key
F10	F10 key
F11	F11 key
F12	F12 key
F13	F13 key
F14	F14 key

Symbolic Name	Mouse or Keyboard Equivalent
F15	F15 key
F16	F16 key
F17	F17 key
F18	F18 key
F19	F19 key
F20	F20 key
F21	F21 key
F22	F22 key
F23	F23 key
F24	F24 key
NUMLOCK	NUM LOCK key
SCROLL	SCROLL LOCK key

Appendix A. Notices

Any reference to a non-open source product, program, or service is not intended to state or imply that only non-open source product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any Rexx Language Association (RexxLA) intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-open source product, program, or service.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-open source products was obtained from the suppliers of those products, their published announcements or other publicly available sources. RexxLA has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-RexxLA packages. Questions on the capabilities of non-RexxLA packages should be addressed to the suppliers of those products.

All statements regarding RexxLA's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

A.1. Trademarks

Open Object Rexx™ and ooRexx™ are trademarks of the Rexx Language Association.

The following terms are trademarks of the IBM Corporation in the United States, other countries, or both:

1-2-3
AIX
IBM
Lotus
OS/2
S/390
VisualAge

AMD is a trademark of Advance Micro Devices, Inc.

Intel, Intel Inside (logos), MMX and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the Unites States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in

the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

A.2. Source Code For This Document

The source code for this document is available under the terms of the Common Public License v1.0 which accompanies this distribution and is available in the appendix *Common Public License Version 1.0*. The source code itself is available at `http://sourceforge.net/project/showfiles.php?group_id=119701`.

The source code for this document is maintained in DocBook SGML/XML format.

Appendix B. Common Public License Version 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS COMMON PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

B.1. Definitions

"Contribution" means:

1. in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
2. in the case of each subsequent Contributor:
   a. changes to the Program, and
   b. additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents " mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

B.2. Grant of Rights

1. Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.
2. Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such

combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

3. Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.
4. Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

B.3. Requirements

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

1. it complies with the terms and conditions of this Agreement; and
2. its license agreement:
   a. effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;
   b. effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;
   c. states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and
   d. states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

1. it must be made available under this Agreement; and
2. a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

B.4. Commercial Distribution

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

B.5. No Warranty

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

B.6. Disclaimer of Liability

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE

PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

B.7. General

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed. In addition, if Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. IBM is the initial Agreement Steward. IBM may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

Index

A

AbsRect2LogRect, 181
abStop, 154
Add
  ComboBox class, 525
  ListBox class, 512
  ListControl class, 466
  TreeControl class, 576
Add... methods, 223
AddAttribute, 65
AddAutoStartMethod, 139
AddBitmapButton, 227
AddBlackFrame, 254
AddBlackRect, 253
AddButton, 225
AddButtonGroup, 250
AddCategoryComboEntry, 289
AddCategoryListEntry, 291
AddCheckBox, 239
AddCheckBoxStem, 248
AddCheckGroup, 242
AddComboBox, 238
AddComboEntry, 78
AddComboInput, 246
AddDirectory
  ComboBox class, 531
  ListBox class, 521
AddEntryLine, 233
AddExtendedStyle, 439
AddFullSeq, 562
AddGrayFrame, 253
AddGrayRect, 253
AddGroupBox, 231
AddIcon, 231
AddInput, 243
AddInputGroup, 244
AddInputStem, 247
AddLine
  InputBox class, 304
  PasswordBox class, 305
AddListBox, 237
AddListControl, 396
AddListEntry, 86
AddMenuItem, 257
AddMenuSeparator, 258
AddOkCancelLeftBottom, 255
AddOkCancelLeftTop, 255
AddOkCancelRightBottom, 254
AddOkCancelRightTop, 255
AddPasswordLine, 237
AddPopupMenu, 257
AddProgressBar, 399
AddRadioButton, 240
AddRadioGroup, 241
AddRadioStem, 249
AddRow, 467
AddScrollBar, 249
AddSequence, 562
AddSliderControl, 400
AddStyle
  EditControl class, 426
  ListControl class, 438
AddTabControl, 402
AddText, 232
AddTreeControl, 393
AddUserMsg, 59
AddWhiteFrame, 253
AddWhiteRect, 252
AdjustToRectangle, 570
AdvancedControls class, 379
AlignLeft, 486
AlignTop, 486
animated button methods, 139
AnimatedButton class, 341
appearance methods
  BaseDialog class, 104
Arrange, 485
AskDialog, 318
AssignFocus, 162
AssignWindow, 168
AsyncMessageHandling, 33
AutoDetection, 26

B

B
  DlgArea class, 330
BackgroundBitmap, 132
BackgroundColor, 104
BaseDialog class, 15
bitmap methods, 129, 191
BkColor, 490
BkColor=, 491
Bottom
  DlgArea class, 330
ButtonControl class, 495

C

Cancel, 77
Cancel push buttons, 254
CaptureMouse, 189
CategoryComboAddDirectory, 291
CategoryComboDrop, 291
CategoryDialog class, 273
CategoryListAddDirectory, 293
CategoryListDrop, 293
CategoryPage, 280
Center, 121
ChangeBitmap, 499
ChangeBitmapButton, 129
ChangeCategoryComboEntry, 291
ChangeCategoryListEntry, 293
ChangeComboEntry, 83
ChangeListEntry, 91
ChangePage, 282
Check, 508
  ListControl class, 450
CheckAll, 452
CheckBox class, 509
CheckList
  class, 309
  function, 326
CheckMenuItem, 143
Child, 585
Clear, 178
ClearButtonRect, 128
ClearMessages, 34
ClearRect
  BaseDialog class, 127
  DialogControl class, 178
ClearSelRange, 557
ClearTicks, 550
ClearWindowRect, 128
ClientToScreen, 182
CloseDropDown, 532
Collapse, 589
CollapseAndReset, 590
ColumnInfo, 458
ColumnWidth, 459
ColumnWidth=, 523
CombineELwithSB, 95
combo box methods, 78, 289
ComboAddDirectory, 83
ComboBox class, 525
ComboDrop, 85
Common Public License, 611
connect attribute methods
  BaseDialog class, 61
connect event methods
  BaseDialog class, 35
  DialogControl class, 169
Connect... methods, 283
ConnectAllSBEvents, 58
ConnectAnimatedButton, 140
ConnectBitmapButton, 51
ConnectButton, 50
ConnectButtonNotify, 362
ConnectCheckBox, 63
ConnectComboBox, 62
ConnectComboBoxNotify, 368
ConnectCommonNotify, 345
ConnectControl, 53
ConnectDraw, 54
ConnectEditNotify, 364
ConnectEntryLine, 61
ConnectFKeyPress
  BaseDialog class, 46
  DialogControl class, 172
ConnectHelp, 39
ConnectKeyPress
  BaseDialog class, 41
  DialogControl class, 169
ConnectList, 55
ConnectListBox, 64
ConnectListBoxNotify, 366
ConnectListControl, 392
ConnectListLeftDoubleClick, 55

ConnectListNotify, 352
ConnectListViewNotify, 357
ConnectMenuItem, 142
ConnectMouseCapture, 38
ConnectMove, 37
ConnectMultiListBox, 64
ConnectPosChanged, 38
ConnectRadioButton, 63
ConnectResize, 35
ConnectScrollBar, 56
ConnectScrollBarNotify, 371
ConnectSliderControl, 393
ConnectSliderNotify, 375
ConnectTabControl, 393
ConnectTabNotify, 373
ConnectTreeControl, 392
ConnectTreeNotify, 347
conversion methods, 180
CorrectionFactor
  DlgAreaU class, 337
CountTicks, 551
CPL, 611
Create, 217
CreateBrush, 203
CreateCategoryDialog, 280
CreateCenter, 219
CreateFont, 200
CreateMenu, 256
CreatePen, 204
CurrentCategory, 281
CursorPos, 185
Cursor_AppStarting, 187
Cursor_Arrow, 187
Cursor_Cross, 188
Cursor_No, 188
Cursor_Wait, 189
CX
  DlgArea class, 331
CY
  DlgArea class, 331

D

debugging methods, 145
Define
  TimedMessage, 302
DefineDialog
  CategoryDialog class, 279
  InputBox class, 304
  UserDialog class, 219
DefListDragHandler, 355
DefTreeDragHandler, 350
DeInstall, 78
Delete
  ComboBox class, 526
  ListBox class, 513
  ListControl class, 468
  TabControl class, 565
  TreeControl class, 588
DeleteAll
  ComboBox class, 527
  ListBox class, 513
  ListControl class, 468
  TabControl class, 565
  TreeControl class, 589
DeleteCategoryComboEntry, 289
DeleteCategoryListEntry, 292
DeleteColumn, 456
DeleteComboEntry, 80
DeleteFont, 201
DeleteListEntry, 87
DeleteObject, 205
Deselect, 473
DeselectIndex, 515
DeselectRange, 517
DeterminePosition, 537
DetermineSBPosition, 96
device context methods
  BaseDialog class, 133
  DialogControl class, 192
dialog control methods, 255
DialogControl class, 147
DimBitmap, 505
Disable, 153
disable methods
  BaseDialog class, 104
  CategoryDialog class, 294
  DialogControl class, 150
DisableCategoryItem, 294
DisableItem, 114
DisableMenuItem, 143
DisconnectKeyPress
  BaseDialog class, 47
  DialogControl class, 174
DisplaceBitmap

BaseDialog class, 132
ButtonControl class, 501
Display, 152
DlgArea class, 329
DlgArea, DlgAreaU classes, 329
DlgAreaU class, 336
DlgUtil class, 298
Draw, 177
draw methods
BaseDialog class, 125
DialogControl class, 177
DrawAngleArc, 211
DrawArc, 208
DrawBitmap
BaseDialog class, 130
ButtonControl class, 504
DrawButton, 125
DrawLine, 206
DrawPie, 209
DrawPixel, 207
DropHighlight, 597
DropHighlighted
ListControl class, 475
TreeControl class, 586
Dump, 146
Dynamically Resizable dialogs
DlgAreaU class, 338

E

Edit
ListControl class, 488
TreeControl class, 594
EditControl class, 407
EditSelection, 532
Enable, 152
enable methods
BaseDialog class, 104
CategoryDialog class, 294
DialogControl class, 150
EnableCategoryItem, 294
EnableItem, 114
EnableMenuItem, 143
EndAsyncExecution, 29
EndEdit
ListControl class, 489
TreeControl class, 594
EnsureCaretVisibility, 411
EnsureVisible
BaseDialog class, 106
ListControl class, 480
TreeControl class, 592
ErrorDialog, 318
ErrorMessage, 313
Execute
BaseDialog class, 27
InputBox class, 304
TimedMessage class, 303
ExecuteAsync, 28
Expand, 591
external functions, 312
ErrorMessage, 313
GetFileNameWindow, 314
GetScreenSize, 313
GetSysMetrics, 314
InfoMessage, 313
PlaySoundFile, 314
PlaySoundFileInLoop, 314
SleepMS, 316
StopSoundFile, 314
WinTime, 316
YesNoMessage, 313

F

FilenameDialog, 319
FileViewer.rex
example code, 121
FillDrawing, 210
Find
ComboBox class, 527
ListBox class, 513
ListControl class, 483
FindCategoryComboEntry, 290
FindCategoryListEntry, 292
FindComboEntry, 80
FindListEntry, 88
FindNearestXY, 485
FindPartial, 484
FindWindow, 321
FirstVisible
ListControl class, 475
TreeControl class, 586
FirstVisibleLine, 411

Focus
ListControl class, 475
TabControl class, 567
FocusCategoryItem, 295
Focused
ListControl class, 474
TabControl class, 568
FocusItem, 111
FontColor, 202
FontToDC, 201
ForegroundWindow, 158
Frames, 251
FreeButtonDC, 135
FreeDC, 192
FreeWindowDC, 134

G

Get, 97, 98
get and set methods
BaseDialog class, 66
get methods
CategoryDialog class, 285
GetArcDirection, 209
GetAttrib, 74
GetBitmapSizeX
BaseDialog class, 129
ButtonControl class, 503
GetBitmapSizeY
BaseDialog class, 130
ButtonControl class, 504
GetBmpDisplacement
BaseDialog class, 133
ButtonControl class, 501
GetButtonControl, 381
GetButtonDC, 134
GetButtonRect, 101
GetCategoryAttrib, 288
GetCategoryCheckBox, 287
GetCategoryComboEntry, 290
GetCategoryComboItems, 290
GetCategoryComboLine, 287
GetCategoryEntryLine, 285
GetCategoryListEntry, 292
GetCategoryListItems, 292
GetCategoryListLine, 286
GetCategoryListWidth, 286
GetCategoryMultiList, 286
GetCategoryRadioButton, 287
GetCategoryValue, 288
Getcheck, 453
GetCheckBox, 73
GetCheckControl, 383
GetClientRect, 161
GetComboBox, 385
GetComboEntry, 81
GetComboItems, 81
GetComboLine, 71
GetControlID, 99
GetCurrentCategoryComboIndex, 290
GetCurrentCategoryListIndex, 292
GetCurrentComboIndex, 81
GetCurrentListIndex, 90
GetData, 66
GetDataStem, 76
GetDC, 192
GetEditControl, 381
GetEntryLine, 68
GetExtendedStyle, 445
GetExtendedStyleRaw, 447
GetFileNameWindow, 314
GetFirstVisible, 519
GetFocus, 163
GetHoverTime, 448
GetID, 158
GetItem, 99
GetLine, 415
GetLineStep, 553
GetListBox, 384
GetListControl, 388
GetListEntry, 88
GetListItemHeight, 89
GetListItems, 89
GetListLine, 69
GetListWidth, 85
GetMenuItemState, 145
GetMouseCapture, 189
GetMultiList, 70
GetPageStep, 553
GetPixel, 207
GetPos
BaseDialog class, 101
DialogControl class, 162
GetProgressBar, 389
GetRadioButton, 72

GetRadioControl, 382
GetRect, 160
GetSBPos, 94
GetSBRange, 93
GetScreenSize, 313
GetScrollBar, 386
GetSelectedPage, 281
GetSize, 162
GetSliderControl, 390
GetStaticControl, 380
GetStyle
  EditControl class, 430
GetStyleRaw, 158
GetSysMetrics, 314
GetSystemMetrics, 102
GetTabControl, 391
GetText, 416
  ComboBox class, 530
  ListBox class, 519
GetTextSize, 199
GetTick, 551
GetTreeControl, 387
GetValue, 73
GetWindowDC, 134
GetWindowRect, 102
graphic drawing methods, 205
graphic methods, 203
GrayMenuItem, 144

H

H
  DlgArea class, 331
HandleMessages, 33
HasKeyPressConnection
  BaseDialog class, 49
  DialogControl class, 176
Help, 77
Hide
  DialogControl class, 151
hide methods
  BaseDialog class, 104
  CategoryDialog class, 294
  DialogControl class, 150
HideCategoryItem, 294
HideFast, 151
HideItem, 114
HideItemFast, 115
HideWindow, 116
HideWindowFast, 116
HitTest, 599
HR
  DlgArea class, 331
HScrollPos, 184

I

Indent, 593
Indent=, 593
Indeterminate, 508
InfoDialog, 318
InfoMessage, 313
Init
  CategoryDialog class, 275
  CheckList class, 309
  DlgArea class, 330
  DlgAreaU class, 336
  InputBox class, 303
  ListChoice class, 307
  MultiInputBox class, 306
  RcDialog class, 270
  ResDialog class, 267
  SingleList class, 311
  TimedMessage class, 301
  UserDialog class, 216
InitAutoDetection
  BaseDialog class, 25
  UserDialog class, 217
InitCategories, 277
InitDialog
  BaseDialog class, 26
  CategoryDialog class, 280
InitRange, 547
InitSelRange, 555
InputBox
  class, 303
  function, 325
Insert
  ComboBox class, 526
  ListBox class, 512
  ListControl class, 461
  TabControl class, 560
  TreeControl class, 574
InsertCategoryComboEntry, 289

InsertCategoryListEntry, 291
InsertColumn, 455
InsertComboEntry, 79
InsertListEntry, 87
IntegerBox
  class, 305
  function, 325
IsAncestor, 601
IsChecked, 507
  ListControl class, 454
IsDialogActive, 32
IsDropDownOpen, 532
IsMaximized, 110
IsMinimized, 109
IsModified, 411
IsMouseButtonDown, 190
ItemHeight, 523
ItemHeight=, 523
ItemInfo
  ListControl class, 470
  TabControl class, 564
  TreeControl class, 581
ItemPos, 487
Items
  ComboBox class, 529
  ListBox class, 517
  ListControl class, 469
  TabControl class, 563
  TreeControl class, 583
ItemsPerPage, 489
ItemState, 472
ItemText, 471
ItemTitle, 67

K

KeyName, 603

L

L
  DlgArea class, 332
Last
  ListControl class, 469
  TabControl class, 566
LastError
  DlgAreaU class, 337
LastSelected, 474
Leaving, 78
Left
  DlgArea class, 332
License, Common Public, 611
License, Open Object Rexx, 611
LineFromIndex, 414
LineIndex, 413
LineLength, 414
Lines, 412
LineScroll, 410
list box methods, 85, 291
ListAddDirectory, 92
ListBox class, 511
ListChoice
  class, 307
  function, 326
ListControl class, 433
ListDrop, 92
Load, 220
LoadBitmap, 191
LoadFrame, 221
LoadItems, 222
LoadMenu, 258
LogRect2AbsRect, 180

M

MakeFirstVisible
  ListBox class, 518
  TreeControl class, 596
Margim
  DlgArea class, 332
Margins, 420
Maximize, 108
Menu methods, 256
  BaseDialog class, 142
MessageExtensions class, 345
methods for dialog items, 284
methods for handles, sizes, and positions, 97
methods to query OS values, 97
Minimize, 107
miscellaneous dialog control methods, 166
modification messages
  ListControl class, 493

Modify
ComboBox class, 530
ListBox class, 520
ListControl class, 462
TabControl class, 561
TreeControl class, 579
ModifyColumn, 457
mouse and cursor methods, 185
Move, 156
MoveCategoryItem, 295
MoveItem
BaseDialog class, 120
TreeControl class, 601
MSSleep, 323
MultiInputBox
class, 305
function, 325
MultiListChoice
class, 308
function, 326

N

Next
ListControl class, 476
TreeControl class, 586
NextLeft, 477
NextPage, 282
NextRight, 478
NextSelected, 476
NextVisible, 587
NoAutoDetection, 26
NoMove
DlgAreaU class, 337
NoResize
DlgAreaU class, 337
Notices, 609
notification messages
TreeControl class, 602

O

ObjectToDC, 204
OK, 76
Ok push buttons, 254
ooDialog public routines, 316
ooRexx License, 611
OpaqueText, 196
Open Object Rexx License, 611
OpenDropDown, 532

P

PageHasChanged, 282
Parent, 584
PasswordBox
class, 304
function, 325
PasswordChar, 419
PasswordChar=, 418
PeekDialogMessage, 34
PlainUserDialog class, 261
Play, 317
PlaySoundFile, 314
PlaySoundFileInLoop, 314
Popup, 30
PopupAsChild, 31
Pos, 547
Pos=, 546
Position, 537
PosRectangle, 570
Prepare4nItems, 469
preparing and running dialogs, 24
Previous
ListControl class, 477
TreeControl class, 587
PreviousPage, 282
PreviousSelected, 476
PreviousVisible, 588
ProcessMessage, 166
ProgressBar class, 541
PropertySheet class, 539
public routines
AskDialog, 318
CheckList, 326
ErrorDialog, 318
FileNameDialog, 319
FindWindow, 321
InfoDialog, 318
InputBox, 325
IntegerBox, 325
ListChoice, 326

MSSleep, 323
MultiInputBox, 325
MultiListChoice, 326
PasswordBox, 325
Play, 317
ScreenSize, 322
SingleSelection, 328
SystemMetrics, 322
TimedMessage, 324

R

R
DlgArea class, 332
RadioButton class, 507
Range
ScrollBar class, 536
SliderControl class, 550
RcDialog class, 269
Rectangle, 206
Rectangles, 251
Redraw, 179
RedrawButton, 126
RedrawClient, 180
RedrawItems, 479
RedrawRect
BaseDialog class, 125
DialogControl class, 179
RedrawWindow, 118
RedrawWindowRect, 127
ReleaseMouseCapture, 190
RemoveBitmap, 191
RemoveExtendedStyle, 443
RemoveImages
ListControl class, 483
TabControl class, 569
TreeControl class, 599
RemoveSmallImages, 483
RemoveStyle
EditControl class, 428
ListControl class, 438
ReplaceExtendedStyle, 444
ReplaceSelText, 418
ReplaceStyle
EditControl class, 429
ListControl class, 436
RequiredWindowSize, 571
ResDialog class, 267
Resize, 155
ResizeCategoryItem, 295
ResizeItem, 119
Resource file dialogs, 267
RcDialog class, 269
ResDialog class, 267
Restore, 109
RestoreCursorShape, 187
RestoreEditClass
ListControl class, 489
TreeControl class, 595
Right
DlgArea class, 332
Root, 584
Rows, 564
Run, 27

S

ScreenSize, 322
ScreenToClient, 182
Scroll
ButtonControl class, 502
DialogControl class, 183
ListControl class, 490
scroll bar methods, 93
scroll methods, 183
ScrollBar class, 535
ScrollBitmapFromTo
BaseDialog class, 131
ButtonControl class, 506
ScrollButton, 138
ScrollCommand, 409
ScrollInButton, 137
ScrollText
BaseDialog class, 136
ButtonControl class, 502
Select
ComboBox class, 529
EditControl class, 408
ListBox class, 515
ListControl class, 473
TabControl class, 566
TreeControl class, 595
Selected
ComboBox class, 528

EditControl class, 408
ListBox class, 514
ListControl class, 474
TabControl class, 566
TreeControl class, 585
SelectedIndex
ComboBox class, 528
ListBox class, 514
TabControl class, 566
SelectedIndexes, 518
SelectedItems
ListBox class, 518
ListControl class, 470
SelectIndex
ComboBox class, 529
ListBox class, 515
TabControl class, 567
SelectRange, 516
SelRange, 558
SendMessageToCategoryItem, 295
SendMessageToItem, 34
set methods
CategoryDialog class, 285
SetArcDirection, 209
SetAttrib, 75
SetCategoryAttrib, 288
SetCategoryCheckBox, 288
SetCategoryComboLine, 287
SetCategoryEntryLine, 285
SetCategoryItemFont, 294
SetCategoryListLine, 286
SetCategoryListTabulators, 293
SetCategoryListWidth, 286
SetCategoryMultiList, 287
SetCategoryRadioButton, 287
SetCategoryStaticText, 285
SetCategoryValue, 288
SetCheckBox, 73
SetColor, 164
SetColumnWidth, 460
SetComboLine, 71
SetCurrentCategoryComboIndex, 290
SetCurrentCategoryListIndex, 293
SetCurrentComboIndex, 82
SetCurrentListIndex, 90
SetCursorPos, 186
SetData, 67
SetDataStem, 75
SetEntryLine, 68
SetFocus, 163
SetFocusToWindow, 163
SetFont, 200
SetForegroundWindow, 157
SetGroup, 112, 153
SetHoverTime, 449
SetHScrollPos, 184
SetImages
ListControl class, 482
TabControl class, 568
TreeControl class, 598
SetItemFont, 138
SetItemPos, 487
SetItemState, 464
SetItemText, 463
SetLimit, 418
SetLineStep, 554
SetListColumnWidth, 86
SetListItemHeight, 90
SetListLine, 69
SetListTabulators, 91
SetListWidth, 85
SetMargins, 420
SetMax, 549
SetMenu
ResDialog class, 269
UserDialog class, 258
SetMenuItemRadio, 144
SetMin, 548
SetModified, 412
SetMultiList, 71
SetPadding, 569
SetPageStep, 554
SetPos
ProgressBar class, 542
ScrollBar class, 536
SliderControl class, 546
SetRadioButton, 72
SetRange
ProgressBar class, 543
ScrollBar class, 535
SetReadOnly, 419
SetRect, 160
SetSBPos, 94
SetSBRange, 93
SetSelEnd, 557
SetSelStart, 556

SetSize, 569
SetSmallImages, 481
SetStaticText, 68
SetStep, 542
SetTabStop, 113
SetTabulators, 520
SetText, 416
SetTickAt, 552
SetTickFrequency, 552
Setting up the dialog, 275
SetTitle, 166
SetValue, 74
SetVScrollPos, 185
SetWidth, 522
SetWindowRect, 117
SetWindowTitle, 121
Show
  BaseDialog class, 104
  DialogControl class, 150
show methods
  BaseDialog class, 104
  CategoryDialog class, 294
  DialogControl class, 150
ShowBalloon, 421, 422, 425
ShowCategoryItem, 294
ShowFast, 151
ShowItem, 115
ShowItemFast, 116
ShowWindow, 117
ShowWindowFast, 117
SingleSelection
  class, 311
  function, 328
SleepMS, 316
SliderControl class, 545
SmallSpacing, 478
SnapToGrid, 486
SortChildren, 597
Spacing, 478
standard dialogs, 299
  CheckList, 309
  InputBox, 303
  IntegerBox, 305
  ListChoice, 307
  MultiInputBox, 305
  MultiListChoice, 308
  PasswordBox, 304
  SingleSelection, 311
  TimedMessage, 300
Standard Dialogs, External Functions, Public Routines, 297
standard event methods, 76
StartIt
  CategoryDialog class, 283
  ResDialog class, 268
  UserDialog class, 255
State, 495
State=, 496
StaticControl class, 405
Step, 541
StopIt
  BaseDialog class, 33
  UserDialog class, 256
StopSoundFile, 314
StringWidth, 461
Style=, 498
SubclassEdit
  ListControl class, 489
  TreeControl class, 595
symbolic names for virtual keys, 604
SystemMetrics, 322

T

T
  DlgArea class, 332
TabControl class, 559
TabToNext, 111
TabToPrevious, 111
text methods
  BaseDialog class, 135
  DialogControl class, 193
TextBkColor, 492
TextBkColor=, 492
TextColor, 491
TextColor=, 491
TiledBackgroundBitmap, 131
TimedMessage
  class, 300
  function, 324
Title, 165
Title=, 165
Toggle, 592
Top
  DlgArea class, 332

ToTheTop, 105
TransparentText, 195
TreeControl class, 573

U

Uncheck, 508
  ListControl class, 451
UncheckAll, 452
UncheckMenuItem, 144
Update
  DialogControl class, 165
  ListControl class, 480
UpdateItem, 479
UserDialog class, 213

V

Validate
  BaseDialog class, 78
  IntegerBox function, 305
Value, 167
Value=, 167
VCode, 603
version, 298
view styles, 436
VirtualKeyCodes class, 603
VisibleItems, 583
VScrollPos, 184

W

W
  DlgArea class, 333
Width, 522
WinTimer, 316
WR
  DlgArea class, 333
Write
  BaseDialog class, 135
  DialogControl class, 193
WriteDirect, 195
WriteToButton, 198
WriteToWindow, 196

X

X
  DlgArea class, 333

Y

Y
  DlgArea class, 334
YesNoMessage, 313