

Dyalog for Microsoft Windows Object Reference Guide

Dyalog version 20.0



DYALOG

*Dyalog is a trademark of Dyalog Limited
Copyright © 1982-2025 by Dyalog Limited
All rights reserved.*

Dyalog for Microsoft Windows Object Reference Guide

Dyalog version: 20.0

Document Revision: 2025-10-30 main:e0843eae32

Unless stated otherwise, all examples in this document assume that □IO □ML ← 1

Dyalog Limited makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Dyalog Limited reserves the right to revise this publication without notification.

email: support@dyalog.com

<https://www.dyalog.com>

TRADEMARKS:

Raspberry Pi is a trademark of the Raspberry Pi Foundation.

Oracle®, MySQL, and Java™ are registered trademarks of Oracle and/or its affiliates.

JavaScript™ is a trademark of Oracle Corporation.

Unicode is a registered trademarks of Unicode, Inc. in the U.S. and other countries.

UNIX® is a registered trademark in the U.S. and other countries, licensed exclusively through X/Open Company Limited.

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Windows® is a registered trademark of Microsoft Corporation in the U.S. and other countries.

macOS® and OS X® (operating system software) are registered trademarks of Apple Inc. in the U.S. and other countries.

All other trademarks and copyrights are acknowledged.

Except where otherwise noted, this content is licensed under a [Creative Commons Attribution 4.0 International](https://creativecommons.org/licenses/by/4.0/) licence.

Contents

1 GUI Overview	44
1.1 GUI Overview	44
1.2 Objects Categorised	45
1.3 Objects A-Z	49
1.4 Properties A-Z	53
1.5 Events A-Z	69
1.6 Methods A-Z	78
2 Objects	84
2.1 ActiveXContainer Object	84
2.2 ActiveXControl Object	85
2.3 Animation Object	88
2.4 Bitmap Object	89
2.5 BrowseBox Object	91
2.6 Button Object	92
2.7 ButtonEdit Object	95
2.8 Calendar Object	97
2.9 Circle Object	100
2.10 Clipboard Object	102
2.11 ColorButton Object	104
2.12 Combo Object	106
2.13 ComboEx Object	108
2.14 CoolBand Object	109
2.15 CoolBar Object	111
2.16 Cursor Object	115
2.17 DateTimePicker Object	116
2.18 Edit Object	118
2.19 Ellipse Object	121
2.20 FileBox Object	125

2.21 Font Object	127
2.22 Form Object	129
2.23 Grid Object	131
2.24 Group Object	138
2.25 HTMLRenderer Object	138
2.26 Icon Object	142
2.27 Image Object	144
2.28 ImageList Object	146
2.29 Label Object	147
2.30 List Object	148
2.31 ListView Object	149
2.32 Locator Object	154
2.33 Marker Object	156
2.34 MDIClient Object	159
2.35 Menu Object	160
2.36 MenuBar Object	162
2.37 MenuItem Object	163
2.38 Metafile Object	165
2.39 MsgBox Object	167
2.40 NetClient Object	169
2.41 NetControl Object	170
2.42 NetType Object	173
2.43 OCXClass Object	174
2.44 OLEClient Object	174
2.45 OLEServer Object	175
2.46 Poly Object	177
2.47 Printer Object	180
2.48 ProgressBar Object	182
2.49 PropertyPage Object	183
2.50 PropertySheet Object	187
2.51 Rect Object	189
2.52 RichEdit Object	192
2.53 Root Object	194
2.54 Scroll Object	197
2.55 Separator Object	200
2.56 SM Object	201
2.57 Spinner Object	203

2.58 Splitter Object	205
2.59 Static Object	208
2.60 StatusBar Object	209
2.61 StatusField Object	211
2.62 SubForm Object	213
2.63 SysTrayItem Object	214
2.64 TabBar Object	215
2.65 TabBtn Object	217
2.66 TabButton Object	218
2.67 TabControl Object	219
2.68 TCPSocket Object	222
2.69 Text Object	223
2.70 Timer Object	225
2.71 TipField Object	226
2.72 ToolBar Object	227
2.73 ToolButton Object	229
2.74 ToolControl Object	231
2.75 TrackBar Object	236
2.76 TreeView Object	239
2.77 UpDown Object	241
3 Properties	243
3.1 APLVersion Property	243
3.2 Accelerator Property	244
3.3 AcceptFiles Property	244
3.4 Active Property	244
3.5 Align Property	245
3.6 AlignChar Property	247
3.7 AllowContextMenu Property	247
3.8 AlphaBlend Property	248
3.9 AlwaysShowBorder Property	248
3.10 AlwaysShowSelection Property	248
3.11 ArcMode Property	249
3.12 Array Property	249
3.13 AsChild Property	249
3.14 Attach Property	250
3.15 AutoArrange Property	252
3.16 AutoBrowse Property	252

3.17 AutoConf Property	252
3.18 AutoExpand Property	253
3.19 AutoPlay Property	254
3.20 BCol Property	254
3.21 BandBorders Property	256
3.22 BaseClass Property	256
3.23 Bits Property	257
3.24 Border Property	257
3.25 BrowseFor Property	258
3.26 BtnPix Property	258
3.27 Btns Property	259
3.28 ButtonsAcceptFocus Property	260
3.29 CBits Property	260
3.30 CEFVersion Property	261
3.31 CMap Property	262
3.32 CalendarCols Property	264
3.33 Cancel Property	264
3.34 Caption Property	265
3.35 CaseSensitive Property	265
3.36 CellFonts Property	266
3.37 CellHeights Property	266
3.38 CellSelect Property	266
3.39 CellSet Property	268
3.40 CellTypes Property	269
3.41 CellWidths Property	270
3.42 Changed Property	270
3.43 CharFormat Property	271
3.44 CharSet Property	272
3.45 CheckBoxes Property	274
3.46 Checked Property	275
3.47 ChildEdge Property	275
3.48 ChildList Property	276
3.49 CircleToday Property	276
3.50 ClassID Property	277
3.51 ClassName Property	277
3.52 ClipCells Property	278
3.53 ColLineTypes Property	279

3.54 ColSortImages Property	280
3.55 ColTitle3D Property	281
3.56 ColTitleAlign Property	281
3.57 ColTitleBCol Property	282
3.58 ColTitleDepth Property	282
3.59 ColTitleFCol Property	283
3.60 ColTitles Property	284
3.61 Collate Property	284
3.62 ColorMode Property	284
3.63 ColumnWidth Property	285
3.64 Container Property	285
3.65 Coord Property	285
3.66 Copies Property	287
3.67 CornerTitleBCol Property	287
3.68 Cue Property	288
3.69 CurCell Property	289
3.70 CurrentColor Property	289
3.71 CurrentState Property	290
3.72 CursorObj Property	290
3.73 CustomColors Property	292
3.74 CustomFormat Property	292
3.75 Data Property	294
3.76 DateTime Property	294
3.77 DbClickToggle Property	295
3.78 Decimals Property	295
3.79 Default Property	295
3.80 DefaultColors Property	296
3.81 Depth Property	296
3.82 DevCaps Property	298
3.83 Directory Property	299
3.84 Divider Property	299
3.85 DockChildren Property	300
3.86 DockShowCaption Property	301
3.87 Dockable Property	302
3.88 Docked Property	303
3.89 DragItems Property	303
3.90 Draggable Property	303

3.91 DrawMode Property	304
3.92 Duplex Property	306
3.93 EdgeStyle Property	306
3.94 EditImage Property	307
3.95 EditImageIndent Property	308
3.96 EditLabels Property	308
3.97 Elevated Property	309
3.98 Encoding Property	309
3.99 End Property	311
3.100 EnterReadOnlyCells Property	311
3.101 Event Property	312
3.102 EventList Property	325
3.103 ExportedFns Property	325
3.104 ExportedVars Property	326
3.105 FCol Property	326
3.106 FStyle Property	328
3.107 FieldType Property	328
3.108 File Property	331
3.109 FileMode Property	332
3.110 FillCol Property	332
3.111 Filters Property	333
3.112 FireOnce Property	334
3.113 FirstDay Property	334
3.114 Fixed Property	334
3.115 FixedOrder Property	335
3.116 FlatSeparators Property	335
3.117 FontList Property	336
3.118 FontObj Property	337
3.119 FormatString Property	338
3.120 Formats Property	339
3.121 FullRowSelect Property	339
3.122 GridBCol Property	340
3.123 GridFCol Property	341
3.124 GridLineFCol Property	341
3.125 GridLineWidth Property	342
3.126 GridLines Property	342
3.127 GripperMode Property	343

3.128 HAlign Property	343
3.129 HScroll Property	344
3.130 HTML Property	345
3.131 Handle Property	345
3.132 HasApply Property	346
3.133 HasButtons Property	346
3.134 HasCheckBox Property	347
3.135 HasClearButton Property	347
3.136 HasEdit Property	348
3.137 HasHelp Property	348
3.138 HasLines Property	349
3.139 HasTicks Property	349
3.140 HasToday Property	349
3.141 Header Property	350
3.142 HeaderImageIndex Property	350
3.143 HeaderImageList Property	350
3.144 HelpButton Property	351
3.145 HelpFile Property	351
3.146 HighlightHeaders Property	351
3.147 Hint Property	351
3.148 HintObj Property	352
3.149 HotSpot Property	352
3.150 HotTrack Property	353
3.151 IconObj Property	353
3.152 ImageCount Property	354
3.153 ImageIndex Property	354
3.154 ImageListObj Property	354
3.155 Indents Property	355
3.156 Index Property	355
3.157 Input Property	356
3.158 InputMode Property	358
3.159 InputModeKey Property	360
3.160 InputProperties Property	360
3.161 InstanceMode Property	361
3.162 InterceptedURLs Property	361
3.163 Interval Property	363
3.164 IsLoading Property	363

3.165 Italic Property	363
3.166 ItemGroupMetrics Property	363
3.167 ItemGroups Property	365
3.168 Items Property	367
3.169 Justify Property	368
3.170 KeepBits Property	368
3.171 KeepOnClose Property	369
3.172 LStyle Property	370
3.173 LWidth Property	371
3.174 LastError Property	371
3.175 LateBind Property	371
3.176 LicenseKey Property	372
3.177 Limits Property	372
3.178 LocalAddr Property	372
3.179 LocalAddrName Property	373
3.180 LocalPort Property	373
3.181 LocalPortName Property	374
3.182 Locale Property	375
3.183 MDIActive Property	376
3.184 MDIActiveObject Property	376
3.185 MDIMenu Property	377
3.186 MapCols Property	377
3.187 Mask Property	378
3.188 MaskCol Property	378
3.189 Masked Property	379
3.190 MaxButton Property	380
3.191 MaxDate Property	380
3.192 MaxLength Property	380
3.193 MaxSelCount Property	381
3.194 MetafileObj Property	381
3.195 MethodList Property	382
3.196 MinButton Property	383
3.197 MinDate Property	383
3.198 MonthDelta Property	383
3.199 Moveable Property	384
3.200 MultiColumn Property	384
3.201 MultiLine Property	385

3.202 MultiSelect Property	385
3.203 NewLine Property	385
3.204 Note Property	386
3.205 OLEControls Property	387
3.206 OLEServers Property	387
3.207 OnTop Property	387
3.208 Orientation Property	388
3.209 OtherButton Property	388
3.210 OverflowChar Property	389
3.211 PName Property	390
3.212 PageActive Property	390
3.213 PageActiveObject Property	390
3.214 PageSize Property	391
3.215 PageWidth Property	391
3.216 PaperSize Property	391
3.217 PaperSizes Property	392
3.218 PaperSource Property	392
3.219 PaperSources Property	393
3.220 ParaFormat Property	393
3.221 Password Property	394
3.222 PathWordBreak Property	394
3.223 Picture Property	394
3.224 Points Property	395
3.225 Popup Property	396
3.226 Posn Property	396
3.227 PrintList Property	397
3.228 PrintRange Property	397
3.229 ProgressStyle Property	398
3.230 PropList Property	399
3.231 QueueEvents Property	400
3.232 RTFText Property	401
3.233 Radius Property	401
3.234 RadiusMode Property	402
3.235 Range Property	402
3.236 ReadOnly Property	402
3.237 RealSize Property	403
3.238 Redraw Property	403

3.239 RemoteAddr Property	404
3.240 RemoteAddrName Property	404
3.241 RemotePort Property	405
3.242 RemotePortName Property	405
3.243 ReportBCol Property	406
3.244 ReportImageIndex Property	406
3.245 ReportInfo Property	407
3.246 ResizeColTitles Property	407
3.247 ResizeCols Property	408
3.248 ResizeRowTitles Property	408
3.249 ResizeRows Property	409
3.250 Resolution Property	409
3.251 Resolutions Property	409
3.252 Rotate Property	410
3.253 RowHiddenDepth Property	410
3.254 RowLineTypes Property	411
3.255 RowTitleAlign Property	412
3.256 RowTitleBCol Property	412
3.257 RowTitleDepth Property	413
3.258 RowTitleFCol Property	414
3.259 RowTitles Property	414
3.260 RowTreeDepth Property	415
3.261 RowTreeImages Property	416
3.262 RowTreeStyle Property	418
3.263 Rows Property	420
3.264 RunMode Property	421
3.265 ScrollOpposite Property	422
3.266 SelDate Property	422
3.267 SelImageIndex Property	423
3.268 SelItems Property	423
3.269 SelRange Property	424
3.270 SelText Property	424
3.271 SelectionBorderWidth Property	424
3.272 SelectionColor Property	425
3.273 SelectionColorAlpha Property	425
3.274 ServerVersion Property	425
3.275 ShowCaptions Property	426

3.276 ShowCueWhenFocused Property	426
3.277 ShowDropDown Property	426
3.278 ShowInput Property	427
3.279 ShowSession Property	428
3.280 ShowThumb Property	429
3.281 SingleClickExpand Property	429
3.282 Size Property	429
3.283 Sizeable Property	430
3.284 SocketNumber Property	431
3.285 SocketType Property	431
3.286 SortItems Property	431
3.287 SplitObj1 Property	432
3.288 SplitObj2 Property	432
3.289 Start Property	433
3.290 StartIn Property	433
3.291 State Property	434
3.292 Step Property	434
3.293 Style Property	435
3.294 SysMenu Property	438
3.295 TabFocus Property	438
3.296 TabIndex Property	439
3.297 TabJustify Property	439
3.298 TabObj Property	440
3.299 TabSize Property	441
3.300 Target Property	441
3.301 TargetState Property	441
3.302 Text Property	442
3.303 TextSize Property	443
3.304 Thumb Property	444
3.305 ThumbRect Property	444
3.306 TickAlign Property	445
3.307 TickSpacing Property	445
3.308 Tip Property	445
3.309 TipObj Property	446
3.310 TitleHeight Property	446
3.311 TitleWidth Property	446
3.312 Today Property	447

3.313 ToolboxBitmap Property	447
3.314 TrackRect Property	447
3.315 Translate Property	448
3.316 Transparent Property	449
3.317 Type Property	449
3.318 TypeLibFile Property	449
3.319 TypeLibID Property	449
3.320 TypeList Property	450
3.321 URL Property	450
3.322 Underline Property	451
3.323 UndocksToRoot Property	451
3.324 VAlign Property	451
3.325 VScroll Property	452
3.326 ValidIfEmpty Property	453
3.327 Value Property	453
3.328 Values Property	454
3.329 VariableHeight Property	454
3.330 View Property	454
3.331 Visible Property	456
3.332 WantsReturn Property	457
3.333 WeekNumbers Property	457
3.334 Weight Property	457
3.335 WordFormat Property	458
3.336 Wrap Property	458
3.337 XRange Property	458
3.338 YRange Property	459
3.339 Yield Property	459
4 Events	460
4.1 ActivateApp Event	460
4.2 AddCol Event	460
4.3 AddRow Event	462
4.4 AmbientChanged Event	463
4.5 AnimStarted Event	464
4.6 AnimStopped Event	464
4.7 BadValue Event	465
4.8 BalloonHide Event	465
4.9 BalloonShow Event	466

4.10 BalloonTimeout Event	466
4.11 BalloonUserClick Event	467
4.12 BeginEditLabel Event	467
4.13 CalendarDbClick Event	468
4.14 CalendarDown Event	468
4.15 CalendarMove Event	469
4.16 CalendarUp Event	470
4.17 CellChange Event	471
4.18 CellChanged Event	472
4.19 CellDbClick Event	473
4.20 CellDown Event	474
4.21 CellError Event	475
4.22 CellMove Event	477
4.23 CellOver Event	478
4.24 CellUp Event	479
4.25 Change Event	480
4.26 ClickComment Event	482
4.27 ClipChange Event	483
4.28 Close Event	483
4.29 CloseUp Event	484
4.30 ColorChange Event	484
4.31 ColumnClick Event	485
4.32 Configure Event	486
4.33 ContextMenu Event	488
4.34 Create Event	488
4.35 DDE Event	489
4.36 DateTimeChange Event	490
4.37 DisplayChange Event	491
4.38 DoPopup Event	491
4.39 DockAccept Event	494
4.40 DockCancel Event	494
4.41 DockEnd Event	495
4.42 DockMove Event	495
4.43 DockRequest Event	496
4.44 DockStart Event	497
4.45 DragDrop Event	498
4.46 DropDown Event	498

4.47 DropFiles Event	499
4.48 DropObjects Event	500
4.49 DyalogCustomMessage1 Event	501
4.50 EndEditLabel Event	502
4.51 EndSplit Event	502
4.52 ExitApp Event	503
4.53 ExitWindows Event	504
4.54 Expanding Event	504
4.55 Expose Event	505
4.56 FileBoxCancel Event	506
4.57 FileBoxOK Event	506
4.58 FontCancel Event	506
4.59 FontOK Event	507
4.60 FrameContextMenu Event	507
4.61 GesturePan Event	508
4.62 GesturePressAndTap Event	509
4.63 GestureRotate Event	510
4.64 GestureTwoFingerTap Event	512
4.65 GestureZoom Event	512
4.66 GetDayStates Event	514
4.67 GetTipText Event	516
4.68 GotFocus Event	516
4.69 GridCopy Event	517
4.70 GridCopyError Event	518
4.71 GridCut Event	519
4.72 GridDelete Event	520
4.73 GridDropSel Event	520
4.74 GridKeyPress Event	522
4.75 GridPaste Event	522
4.76 GridPasteError Event	524
4.77 GridSelect Event	524
4.78 HScroll Event	525
4.79 HTTPRequest Event	526
4.80 HThumbDrag Event	530
4.81 Help Event	531
4.82 HideComment Event	531
4.83 Idle Event	532

4.84 IndexChanged Event	532
4.85 ItemDbClick Event	533
4.86 ItemDown Event	533
4.87 ItemUp Event	534
4.88 KeyError Event	535
4.89 KeyPress Event	536
4.90 LoadEnd Event	539
4.91 Locator Event	540
4.92 LostFocus Event	541
4.93 MDIActivate Event	541
4.94 MDIDeactivate Event	542
4.95 MouseDbClick Event	542
4.96 MouseDown Event	543
4.97 MouseEnter Event	544
4.98 MouseLeave Event	545
4.99 MouseMove Event	545
4.100 MouseUp Event	546
4.101 MouseWheel Event	547
4.102 MsgBtn1 Event	548
4.103 MsgBtn2 Event	548
4.104 MsgBtn3 Event	548
4.105 PageActivate Event	549
4.106 PageApply Event	549
4.107 PageBack Event	550
4.108 PageCancel Event	550
4.109 PageChanged Event	551
4.110 PageDeactivate Event	551
4.111 PageFinish Event	552
4.112 PageHelp Event	552
4.113 PageNext Event	552
4.114 PreCreate Event	553
4.115 Protected Event	553
4.116 Retracting Event	554
4.117 Scroll Event	555
4.118 SelDateChange Event	555
4.119 Select Event	556
4.120 SelectCertificate Event	557

4.121 ServiceNotification Event	558
4.122 SetColSize Event	559
4.123 SetItemPosition Event	560
4.124 SetRowSize Event	560
4.125 SetSpinnerText Event	561
4.126 ShowComment Event	562
4.127 Spin Event	563
4.128 Splitting Event	564
4.129 StartSplit Event	564
4.130 StateChange Event	565
4.131 SysColorChange Event	565
4.132 TCPAccept Event	566
4.133 TCPClose Event	567
4.134 TCPConnect Event	567
4.135 TCPErrror Event	568
4.136 TCPGotAddr Event	568
4.137 TCPGotPort Event	569
4.138 TCPReady Event	569
4.139 TCPRecv Event	570
4.140 ThumbDrag Event	571
4.141 Timer Event	571
4.142 VScroll Event	572
4.143 VThumbDrag Event	572
4.144 WebSocketError Event	573
4.145 WebSocketReceive Event	573
4.146 WebSocketUpgrade Event	574
4.147 WinIniChange Event	575
5 Methods	577
5.1 Abort Method	577
5.2 AddChildren Method	577
5.3 AddComment Method	578
5.4 AddItems Method	579
5.5 AnimClose Method	580
5.6 AnimOpen Method	580
5.7 AnimPlay Method	581
5.8 AnimStop Method	581
5.9 Animate Method	582

5.10 Browse Method	583
5.11 CancelToClose Method	584
5.12 CellFromPoint Method	584
5.13 ChooseFont Method	584
5.14 ColChange Method	585
5.15 ColSorted Method	586
5.16 DateToIDN Method	588
5.17 DelCol Method	589
5.18 DelComment Method	589
5.19 DelRow Method	590
5.20 DeleteChildren Method	590
5.21 DeleteItems Method	591
5.22 DeleteTypeLib Method	591
5.23 Detach Method	592
5.24 DuplicateColumn Method	592
5.25 DuplicateRow Method	593
5.26 ExecuteJavaScript Method	593
5.27 FileRead Method	594
5.28 FileWrite Method	594
5.29 Flush Method	595
5.30 GetBuildID Method	595
5.31 GetCellRect Method	596
5.32 GetCommandLine Method	596
5.33 GetCommandLineArgs Method	597
5.34 GetComment Method	597
5.35 GetEnvironment Method	598
5.36 GetEventInfo Method	599
5.37 GetFocus Method	600
5.38 GetFocusObj Method	600
5.39 GetItemHandle Method	601
5.40 GetItemPosition Method	601
5.41 GetItemState Method	602
5.42 GetMethodInfo Method	602
5.43 GetMinSize Method	603
5.44 GetParentItem Method	604
5.45 GetPropertyInfo Method	604
5.46 GetServiceState Method	605

5.47 GetTextSize Method	606
5.48 GetTypeInfo Method	607
5.49 GetVisibleRange Method	608
5.50 GetZoomLevel Method	608
5.51 GreetBitmap Method	608
5.52 IDNToDate Method	609
5.53 ListTypeLibs Method	609
5.54 LockColumns Method	610
5.55 LockRows Method	612
5.56 MDIArrange Method	613
5.57 MDICascade Method	614
5.58 MDITile Method	614
5.59 MakeGIF Method	614
5.60 MakePNG Method	615
5.61 NameFromHandle Method	615
5.62 NewPage Method	616
5.63 OLEAddEventSink Method	616
5.64 OLEDeleteEventSink Method	616
5.65 OLEListEventSinks Method	617
5.66 OLEQueryInterface Method	617
5.67 OLERegister Method	618
5.68 OLEUnregister Method	618
5.69 Print Method	619
5.70 PrintToPDF Method	619
5.71 ProgressStep Method	619
5.72 RTFPrint Method	620
5.73 RTFPrintSetup Method	621
5.74 RowChange Method	622
5.75 RowSetVisibleDepth Method	623
5.76 SetCellSet Method	625
5.77 SetCellType Method	625
5.78 SetEventInfo Method	626
5.79 SetFinishText Method	628
5.80 SetFnInfo Method	629
5.81 SetItemImage Method	631
5.82 SetItemState Method	631
5.83 SetMethodInfo Method	632

5.84 SetPropertyInfo Method	634
5.85 SetServiceState Method	634
5.86 SetVarInfo Method	635
5.87 SetWizard Method	636
5.88 SetZoomLevel Method	636
5.89 Setup Method	638
5.90 ShowBalloonTip Method	639
5.91 ShowDevTools Method	640
5.92 ShowHelp Method	640
5.93 ShowItem Method	641
5.94 ShowProperties Method	641
5.95 TCPGetHostID Method	642
5.96 TCPSend Method	642
5.97 TCPSendPicture Method	643
5.98 Undo Method	643
5.99 Wait Method	644
5.100 WebSocketClose Method	644
5.101 WebSocketSend Method	645
6 (A-Z)	647
6.1 Abort Method	647
6.2 Accelerator Property	647
6.3 AcceptFiles Property	648
6.4 ActivateApp Event	648
6.5 Active Property	648
6.6 ActiveXContainer Object	649
6.7 ActiveXControl Object	650
6.8 AddChildren Method	653
6.9 AddCol Event	653
6.10 AddComment Method	655
6.11 AddItems Method	656
6.12 AddRow Event	657
6.13 Align Property	658
6.14 AlignChar Property	660
6.15 AllowContextMenu Property	660
6.16 AlphaBlend Property	661
6.17 AlwaysShowBorder Property	661
6.18 AlwaysShowSelection Property	661

6.19 AmbientChanged Event	662
6.20 Animate Method	663
6.21 Animation Object	664
6.22 AnimClose Method	665
6.23 AnimOpen Method	665
6.24 AnimPlay Method	666
6.25 AnimStarted Event	666
6.26 AnimStop Method	667
6.27 AnimStopped Event	667
6.28 APLVersion Property	668
6.29 ArcMode Property	669
6.30 Array Property	669
6.31 AsChild Property	669
6.32 Attach Property	670
6.33 AutoArrange Property	671
6.34 AutoBrowse Property	671
6.35 AutoConf Property	672
6.36 AutoExpand Property	673
6.37 AutoPlay Property	673
6.38 BadValue Event	674
6.39 BalloonHide Event	674
6.40 BalloonShow Event	675
6.41 BalloonTimeout Event	675
6.42 BalloonUserClick Event	676
6.43 BandBorders Property	676
6.44 BaseClass Property	677
6.45 BCol Property	677
6.46 BeginEditLabel Event	679
6.47 Bitmap Object	679
6.48 Bits Property	682
6.49 Border Property	682
6.50 Browse Method	683
6.51 BrowseBox Object	683
6.52 BrowseFor Property	684
6.53 BtnPix Property	685
6.54 Btns Property	686
6.55 Button Object	687

6.56 ButtonEdit Object	689
6.57 ButtonsAcceptFocus Property	691
6.58 Calendar Object	692
6.59 CalendarCols Property	695
6.60 CalendarDbClick Event	696
6.61 CalendarDown Event	697
6.62 CalendarMove Event	698
6.63 CalendarUp Event	698
6.64 Cancel Property	699
6.65 CancelToClose Method	699
6.66 Caption Property	700
6.67 CaseSensitive Property	700
6.68 CBits Property	701
6.69 CEFVersion Property	701
6.70 CellChange Event	702
6.71 CellChanged Event	703
6.72 CellDbClick Event	704
6.73 CellDown Event	705
6.74 CellError Event	706
6.75 CellFonts Property	708
6.76 CellFromPoint Method	708
6.77 CellHeights Property	708
6.78 CellMove Event	709
6.79 CellOver Event	710
6.80 CellSelect Property	711
6.81 CellSet Property	713
6.82 CellTypes Property	714
6.83 CellUp Event	715
6.84 CellWidths Property	716
6.85 Change Event	716
6.86 Changed Property	718
6.87 CharFormat Property	719
6.88 CharSet Property	720
6.89 CheckBoxes Property	723
6.90 Checked Property	724
6.91 ChildEdge Property	724
6.92 ChildList Property	725

6.93 ChooseFont Method	725
6.94 Circle Object	726
6.95 CircleToday Property	728
6.96 ClassID Property	729
6.97 ClassName Property	729
6.98 ClickComment Event	730
6.99 Clipboard Object	730
6.100 ClipCells Property	732
6.101 ClipChange Event	733
6.102 Close Event	734
6.103 CloseUp Event	734
6.104 CMap Property	735
6.105 ColChange Method	736
6.106 Collate Property	737
6.107 ColLineTypes Property	737
6.108 ColorButton Object	738
6.109 ColorChange Event	739
6.110 ColorMode Property	740
6.111 ColSorted Method	740
6.112 ColSortImages Property	743
6.113 ColTitle3D Property	744
6.114 ColTitleAlign Property	744
6.115 ColTitleBCol Property	745
6.116 ColTitleDepth Property	745
6.117 ColTitleFCol Property	746
6.118 ColTitles Property	747
6.119 ColumnClick Event	747
6.120 ColumnWidth Property	748
6.121 Combo Object	748
6.122 ComboEx Object	750
6.123 Configure Event	752
6.124 Container Property	754
6.125 ContextMenu Event	754
6.126 CoolBand Object	755
6.127 CoolBar Object	757
6.128 Coord Property	761
6.129 Copies Property	762

6.130 CornerTitleBCol Property	763
6.131 Create Event	763
6.132 Cue Property	764
6.133 CurCell Property	765
6.134 CurrentColor Property	765
6.135 CurrentState Property	765
6.136 Cursor Object	766
6.137 CursorObj Property	767
6.138 CustomColors Property	769
6.139 CustomFormat Property	769
6.140 Data Property	771
6.141 DateTime Property	771
6.142 DateTimeChange Event	772
6.143 DateTimePicker Object	772
6.144 DateToIDN Method	774
6.145 DbClickToggle Property	774
6.146 DDE Event	775
6.147 Decimals Property	776
6.148 Default Property	776
6.149 DefaultColors Property	777
6.150 DelCol Method	777
6.151 DelComment Method	778
6.152 DeleteChildren Method	778
6.153 DeleteItems Method	779
6.154 DeleteTypeLib Method	779
6.155 DelRow Method	780
6.156 Depth Property	780
6.157 Detach Method	782
6.158 DevCaps Property	782
6.159 Directory Property	783
6.160 DisplayChange Event	783
6.161 Divider Property	784
6.162 Dockable Property	785
6.163 DockAccept Event	785
6.164 DockCancel Event	786
6.165 DockChildren Property	786
6.166 Docked Property	787

6.167 DockEnd Event	788
6.168 DockMove Event	788
6.169 DockRequest Event	789
6.170 DockShowCaption Property	790
6.171 DockStart Event	791
6.172 DoPopup Event	792
6.173 Dragable Property	795
6.174 DragDrop Event	796
6.175 DragItems Property	797
6.176 DrawMode Property	797
6.177 DropDown Event	798
6.178 DropFiles Event	799
6.179 DropObjects Event	800
6.180 Duplex Property	801
6.181 DuplicateColumn Method	801
6.182 DuplicateRow Method	802
6.183 DyalogCustomMessage1 Event	802
6.184 EdgeStyle Property	803
6.185 Edit Object	804
6.186 EditImage Property	807
6.187 EditImageIndent Property	808
6.188 EditLabels Property	808
6.189 Elevated Property	809
6.190 Ellipse Object	809
6.191 Encoding Property	813
6.192 End Property	815
6.193 EndEditLabel Event	816
6.194 EndSplit Event	817
6.195 EnterReadOnlyCells Property	817
6.196 Event Property	818
6.197 EventList Property	831
6.198 ExecuteJavaScript Method	831
6.199 ExitApp Event	832
6.200 ExitWindows Event	832
6.201 Expanding Event	833
6.202 ExportedFns Property	834
6.203 ExportedVars Property	834

6.204 Expose Event	835
6.205 FCol Property	835
6.206 FieldType Property	837
6.207 File Property	840
6.208 FileBox Object	840
6.209 FileBoxCancel Event	842
6.210 FileBoxOK Event	843
6.211 FileMode Property	843
6.212 FileRead Method	843
6.213 FileWrite Method	844
6.214 FillCol Property	844
6.215 Filters Property	845
6.216 FireOnce Property	846
6.217 FirstDay Property	846
6.218 Fixed Property	846
6.219 FixedOrder Property	847
6.220 FlatSeparators Property	847
6.221 Flush Method	848
6.222 Font Object	848
6.223 FontCancel Event	850
6.224 FontList Property	850
6.225 FontObj Property	851
6.226 FontOK Event	852
6.227 Form Object	852
6.228 Formats Property	855
6.229 FormatString Property	856
6.230 FrameContextMenu Event	857
6.231 FStyle Property	858
6.232 FullRowSelect Property	858
6.233 GesturePan Event	859
6.234 GesturePressAndTap Event	861
6.235 GestureRotate Event	862
6.236 GestureTwoFingerTap Event	863
6.237 GestureZoom Event	864
6.238 GetBuildID Method	865
6.239 GetCellRect Method	866
6.240 GetCommandLine Method	867

6.241 GetCommandLineArgs Method	867
6.242 GetComment Method	868
6.243 GetDayStates Event	868
6.244 GetEnvironment Method	870
6.245 GetEventInfo Method	871
6.246 GetFocus Method	872
6.247 GetFocusObj Method	873
6.248 GetItemHandle Method	873
6.249 GetItemPosition Method	873
6.250 GetItemState Method	874
6.251 GetMethodInfo Method	874
6.252 GetMinSize Method	875
6.253 GetParentItem Method	876
6.254 GetPropertyInfo Method	876
6.255 GetServiceState Method	877
6.256 GetTextSize Method	878
6.257 GetTipText Event	879
6.258 GetTypeInfo Method	880
6.259 GetVisibleRange Method	880
6.260 GetZoomLevel Method	881
6.261 GotFocus Event	881
6.262 GreetBitmap Method	881
6.263 Grid Object	882
6.264 GridBCol Property	888
6.265 GridCopy Event	888
6.266 GridCopyError Event	890
6.267 GridCut Event	890
6.268 GridDelete Event	891
6.269 GridDropSel Event	892
6.270 GridFCol Property	893
6.271 GridKeyPress Event	894
6.272 GridLineFCol Property	895
6.273 GridLines Property	895
6.274 GridLineWidth Property	896
6.275 GridPaste Event	897
6.276 GridPasteError Event	898
6.277 GridSelect Event	899

6.278 GripperMode Property	899
6.279 Group Object	900
6.280 HAlign Property	901
6.281 Handle Property	901
6.282 HasApply Property	902
6.283 HasButtons Property	902
6.284 HasCheckBox Property	903
6.285 HasClearButton Property	903
6.286 HasEdit Property	904
6.287 HasHelp Property	904
6.288 HasLines Property	905
6.289 HasTicks Property	905
6.290 HasToday Property	905
6.291 Header Property	906
6.292 HeaderImageIndex Property	906
6.293 HeaderImageList Property	906
6.294 Help Event	906
6.295 HelpButton Property	907
6.296 HelpFile Property	907
6.297 HideComment Event	908
6.298 HighlightHeaders Property	908
6.299 Hint Property	909
6.300 HintObj Property	909
6.301 HotSpot Property	909
6.302 HotTrack Property	910
6.303 HScroll Property	910
6.304 HScroll Event	911
6.305 HThumbDrag Event	912
6.306 HTML Property	913
6.307 HTMLRenderer Object	913
6.308 HTTPRequest Event	917
6.309 Icon Object	921
6.310 IconObj Property	923
6.311 Idle Event	924
6.312 IDNToDate Method	924
6.313 Image Object	925
6.314 ImageCount Property	926

6.315 ImageIndex Property	927
6.316 ImageList Object	927
6.317 ImageListObj Property	928
6.318 Indents Property	929
6.319 Index Property	929
6.320 IndexChanged Event	930
6.321 Input Property	931
6.322 InputMode Property	933
6.323 InputModeKey Property	934
6.324 InputProperties Property	935
6.325 InstanceMode Property	935
6.326 InterceptedURLs Property	936
6.327 Interval Property	937
6.328 IsLoading Property	937
6.329 Italic Property	938
6.330 ItemDbClick Event	938
6.331 ItemDown Event	938
6.332 ItemGroupMetrics Property	939
6.333 ItemGroups Property	941
6.334 Items Property	943
6.335 ItemUp Event	943
6.336 Justify Property	944
6.337 KeepBits Property	944
6.338 KeepOnClose Property	945
6.339 KeyError Event	946
6.340 KeyPress Event	947
6.341 Label Object	951
6.342 LastError Property	952
6.343 LateBind Property	952
6.344 LicenseKey Property	952
6.345 Limits Property	953
6.346 List Object	953
6.347 ListTypeLibs Method	954
6.348 ListView Object	955
6.349 LoadEnd Event	960
6.350 LocalAddr Property	961
6.351 LocalAddrName Property	961

6.352 Locale Property	962
6.353 LocalPort Property	963
6.354 LocalPortName Property	964
6.355 Locator Object	964
6.356 Locator Event	966
6.357 LockColumns Method	967
6.358 LockRows Method	969
6.359 LostFocus Event	970
6.360 LStyle Property	971
6.361 LWidth Property	972
6.362 MakeGIF Method	972
6.363 MakePNG Method	972
6.364 MapCols Property	973
6.365 Marker Object	973
6.366 Mask Property	976
6.367 MaskCol Property	977
6.368 Masked Property	977
6.369 MaxButton Property	978
6.370 MaxDate Property	979
6.371 MaxLength Property	979
6.372 MaxSelCount Property	979
6.373 MDIActivate Event	980
6.374 MDIActive Property	980
6.375 MDIActiveObject Property	981
6.376 MDIArrange Method	981
6.377 MDICascade Method	981
6.378 MDIClient Object	981
6.379 MDIDeactivate Event	983
6.380 MDIMenu Property	984
6.381 MDITile Method	984
6.382 Menu Object	985
6.383 MenuBar Object	986
6.384 MenuItem Object	988
6.385 Metafile Object	989
6.386 MetafileObj Property	991
6.387 MethodList Property	992
6.388 MinButton Property	993

6.389 MinDate Property	993
6.390 MonthDelta Property	993
6.391 MouseDbClick Event	994
6.392 MouseDown Event	995
6.393 MouseEnter Event	995
6.394 MouseLeave Event	996
6.395 MouseMove Event	996
6.396 MouseUp Event	997
6.397 MouseWheel Event	998
6.398 Moveable Property	999
6.399 MsgBox Object	1000
6.400 MsgBtn1 Event	1002
6.401 MsgBtn2 Event	1003
6.402 MsgBtn3 Event	1003
6.403 MultiColumn Property	1003
6.404 MultiLine Property	1004
6.405 MultiSelect Property	1004
6.406 NameFromHandle Method	1005
6.407 NetClient Object	1005
6.408 NetType Object	1006
6.409 NewLine Property	1007
6.410 NewPage Method	1007
6.411 Note Property	1007
6.412 OCXClass Object	1008
6.413 OLEAddEventSink Method	1009
6.414 OLEClient Object	1009
6.415 OLEControls Property	1010
6.416 OLEDeleteEventSink Method	1010
6.417 OLEListEventSinks Method	1011
6.418 OLEQueryInterface Method	1011
6.419 OLERegister Method	1012
6.420 OLEServer Object	1012
6.421 OLEServers Property	1014
6.422 OLEUnregister Method	1014
6.423 OnTop Property	1014
6.424 Orientation Property	1015
6.425 OtherButton Property	1016

6.426 OverflowChar Property	1016
6.427 PageActivate Event	1017
6.428 PageActive Property	1018
6.429 PageActiveObject Property	1018
6.430 PageApply Event	1018
6.431 PageBack Event	1019
6.432 PageCancel Event	1019
6.433 PageChanged Event	1020
6.434 PageDeactivate Event	1020
6.435 PageFinish Event	1021
6.436 PageHelp Event	1021
6.437 PageNext Event	1021
6.438 PageSize Property	1022
6.439 PageWidth Property	1022
6.440 PaperSize Property	1023
6.441 PaperSizes Property	1023
6.442 PaperSource Property	1023
6.443 PaperSources Property	1024
6.444 ParaFormat Property	1024
6.445 Password Property	1025
6.446 PathWordBreak Property	1025
6.447 Picture Property	1025
6.448 PName Property	1027
6.449 Points Property	1027
6.450 Poly Object	1028
6.451 Popup Property	1031
6.452 Posn Property	1031
6.453 PreCreate Event	1032
6.454 Print Method	1032
6.455 Printer Object	1033
6.456 PrintList Property	1035
6.457 PrintRange Property	1035
6.458 PrintToPDF Method	1036
6.459 ProgressBar Object	1036
6.460 ProgressStep Method	1038
6.461 ProgressStyle Property	1038
6.462 PropertyPage Object	1040

6.463 PropertySheet Object	1043
6.464 PropList Property	1045
6.465 Protected Event	1046
6.466 QueueEvents Property	1046
6.467 Radius Property	1047
6.468 RadiusMode Property	1048
6.469 Range Property	1048
6.470 ReadOnly Property	1048
6.471 RealSize Property	1049
6.472 Rect Object	1049
6.473 Redraw Property	1053
6.474 RemoteAddr Property	1053
6.475 RemoteAddrName Property	1054
6.476 RemotePort Property	1054
6.477 RemotePortName Property	1055
6.478 ReportBCol Property	1055
6.479 ReportImageIndex Property	1056
6.480 ReportInfo Property	1056
6.481 ResizeCols Property	1057
6.482 ResizeColTitles Property	1057
6.483 ResizeRows Property	1058
6.484 ResizeRowTitles Property	1058
6.485 Resolution Property	1059
6.486 Resolutions Property	1059
6.487 Retracting Event	1059
6.488 RichEdit Object	1060
6.489 Root Object	1062
6.490 Rotate Property	1065
6.491 RowChange Method	1065
6.492 RowHiddenDepth Property	1066
6.493 RowLineTypes Property	1067
6.494 Rows Property	1068
6.495 RowSetVisibleDepth Method	1068
6.496 RowTitleAlign Property	1070
6.497 RowTitleBCol Property	1070
6.498 RowTitleDepth Property	1071
6.499 RowTitleFCol Property	1072

6.500 RowTitles Property	1073
6.501 RowTreeDepth Property	1073
6.502 RowTreeImages Property	1074
6.503 RowTreeStyle Property	1076
6.504 RTFPrint Method	1079
6.505 RTFPrintSetup Method	1080
6.506 RTFText Property	1081
6.507 RunMode Property	1082
6.508 Scroll Object	1082
6.509 Scroll Event	1085
6.510 ScrollOpposite Property	1086
6.511 SelDate Property	1087
6.512 SelDateChange Event	1087
6.513 Select Event	1088
6.514 SelectCertificate Event	1089
6.515 SelectionBorderWidth Property	1090
6.516 SelectionColor Property	1090
6.517 SelectionColorAlpha Property	1090
6.518 SellImageIndex Property	1091
6.519 SellItems Property	1091
6.520 SelRange Property	1092
6.521 SelText Property	1092
6.522 Separator Object	1093
6.523 ServerVersion Property	1094
6.524 ServiceNotification Event	1094
6.525 SetCellSet Method	1095
6.526 SetCellType Method	1095
6.527 SetColSize Event	1095
6.528 SetEventInfo Method	1096
6.529 SetFinishText Method	1099
6.530 SetFnInfo Method	1099
6.531 SetItemImage Method	1102
6.532 SetItemPosition Event	1102
6.533 SetItemState Method	1103
6.534 SetMethodInfo Method	1104
6.535 SetPropertyInfo Method	1105
6.536 SetRowSize Event	1106

6.537 SetServiceState Method	1107
6.538 SetSpinnerText Event	1108
6.539 Setup Method	1109
6.540 SetVarInfo Method	1109
6.541 SetWizard Method	1109
6.542 SetZoomLevel Method	1110
6.543 ShowBalloonTip Method	1112
6.544 ShowCaptions Property	1113
6.545 ShowComment Event	1113
6.546 ShowCueWhenFocused Property	1114
6.547 ShowDevTools Method	1115
6.548 ShowDropDown Property	1115
6.549 ShowHelp Method	1116
6.550 ShowInput Property	1117
6.551 ShowItem Method	1118
6.552 ShowProperties Method	1118
6.553 ShowSession Property	1119
6.554 ShowThumb Property	1119
6.555 SingleClickExpand Property	1119
6.556 Size Property	1119
6.557 Sizeable Property	1120
6.558 SM Object	1121
6.559 SocketNumber Property	1123
6.560 SocketType Property	1123
6.561 SortItems Property	1124
6.562 Spin Event	1124
6.563 Spinner Object	1125
6.564 SplitObj1 Property	1127
6.565 SplitObj2 Property	1128
6.566 Splitter Object	1128
6.567 Splitting Event	1131
6.568 Start Property	1132
6.569 StartIn Property	1133
6.570 StartSplit Event	1133
6.571 State Property	1133
6.572 StateChange Event	1134
6.573 Static Object	1135

6.574 StatusBar Object	1136
6.575 StatusField Object	1137
6.576 Step Property	1139
6.577 Style Property	1139
6.578 SubForm Object	1142
6.579 SysColorChange Event	1144
6.580 SysMenu Property	1144
6.581 SysTrayItem Object	1144
6.582 TabBar Object	1145
6.583 TabBtn Object	1147
6.584 TabButton Object	1148
6.585 TabControl Object	1149
6.586 TabFocus Property	1152
6.587 TabIndex Property	1153
6.588 TabJustify Property	1153
6.589 TabObj Property	1154
6.590 TabSize Property	1155
6.591 Target Property	1155
6.592 TargetState Property	1156
6.593 TCPAccept Event	1156
6.594 TCPClose Event	1157
6.595 TCPConnect Event	1158
6.596 TCPError Event	1158
6.597 TCPGetHostID Method	1159
6.598 TCPGotAddr Event	1159
6.599 TCPGotPort Event	1160
6.600 TCPReady Event	1160
6.601 TCPRecv Event	1161
6.602 TCPSend Method	1162
6.603 TCPSendPicture Method	1162
6.604 TCPSocket Object	1163
6.605 Text Object	1164
6.606 Text Property	1167
6.607 TextSize Property	1168
6.608 Thumb Property	1169
6.609 ThumbDrag Event	1169
6.610 ThumbRect Property	1170

6.611 TickAlign Property	1170
6.612 TickSpacing Property	1171
6.613 Timer Object	1171
6.614 Timer Event	1172
6.615 Tip Property	1172
6.616 TipField Object	1173
6.617 TipObj Property	1173
6.618 TitleHeight Property	1174
6.619 TitleWidth Property	1174
6.620 Today Property	1174
6.621 ToolBar Object	1175
6.622 ToolboxBitmap Property	1177
6.623 ToolButton Object	1177
6.624 ToolControl Object	1180
6.625 TrackBar Object	1184
6.626 TrackRect Property	1187
6.627 Translate Property	1187
6.628 Transparent Property	1188
6.629 TreeView Object	1188
6.630 Type Property	1191
6.631 TypeLibFile Property	1191
6.632 TypeLibID Property	1191
6.633 TypeList Property	1192
6.634 Underline Property	1192
6.635 Undo Method	1192
6.636 UndocksToRoot Property	1193
6.637 UpDown Object	1193
6.638 URL Property	1194
6.639 ValidIfEmpty Property	1194
6.640 VAlign Property	1195
6.641 Value Property	1196
6.642 Values Property	1196
6.643 VariableHeight Property	1196
6.644 View Property	1197
6.645 Visible Property	1198
6.646 VScroll Property	1199
6.647 VScroll Event	1200

6.648 VThumbDrag Event	1201
6.649 Wait Method	1201
6.650 WantsReturn Property	1201
6.651 WebSocketClose Method	1202
6.652 WebSocketError Event	1203
6.653 WebSocketReceive Event	1203
6.654 WebSocketSend Method	1204
6.655 WebSocketUpgrade Event	1205
6.656 WeekNumbers Property	1206
6.657 Weight Property	1206
6.658 WinIniChange Event	1206
6.659 WordFormat Property	1207
6.660 Wrap Property	1207
6.661 XRange Property	1208
6.662 Yield Property	1208
6.663 YRange Property	1209
7 Examples	1210
7.1 Calendar	1210
7.2 Calendar	1210
7.3 Calendar	1211
7.4 CoolBar	1212
7.5 CoolBar	1214
7.6 Splitter	1214
7.7 Splitter	1215
7.8 Splitter	1215
7.9 Splitter	1216
7.10 Splitter	1217
7.11 TabControl	1218
7.12 TabControl	1219
7.13 TabControl	1220
7.14 TabControl	1220
7.15 TabControl	1221
7.16 TabControl	1222
7.17 TabControl	1223
7.18 TabControl	1224
7.19 TabControl	1225
7.20 TabControl	1226

7.21 TabControl	1227
7.22 TabControl	1228
7.23 TabControl	1229
7.24 TabControl	1230
7.25 ToolButton	1231
7.26 ToolControl	1232
7.27 ToolControl	1233
7.28 ToolControl	1234
7.29 ToolControl	1234
8 Miscellaneous	1236
8.1 Class Identifier	1236
8.2 COM Data Types	1236
8.3 DISPID	1237
8.4 GUID	1237
8.5 International Day Number	1238
8.6 Native Look and Feel	1238
8.7 Port Number	1239
8.8 Stream Sockets	1239
8.9 Twips	1240
8.10 UDP	1240

1 GUI Overview

1.1 GUI Overview

The Dyalog GUI is based upon the concepts of *objects*, *properties*, *events* and *methods*.

Section 1.2

Objects are instances of classes that contain information and provide functionality. Most Dyalog APL objects are GUI objects that may be displayed on the screen and with which you can interact. An example of an object is a push-button (an instance of class `Button`) which you may press to cause the program to take a particular action. Objects are defined in hierarchies.

Section 1.4

Each object has an associated set of properties which describe how it looks and behaves. For example, a `Button` has a property called `Caption` which defines the character string to be displayed in it.

Section 1.5

During interaction with the user, an object is capable of generating events.

An example is the `Select` event. For a `Button` object, this event is generated when the user presses the `Button`. This can be done in several ways. Firstly, the user may click the left mouse button over the object. Secondly, under certain circumstances, the `Select` event can be generated when the user presses the Enter key. Finally, the event will occur if the user presses a "short-cut" (mnemonic) key that is associated with the `Button`.

Section 1.6

Methods are effectively functions that an object provides; they are things that you may invoke to make the object do something for you. In Dyalog APL, the distinction between methods and events is tenuous, because events also make objects perform actions and you may generate events under program control. For example, a `Scroll` event is

generated by a scrollbar when the user moves the thumb. Conversely, you can make a scrollbar scroll by generating a `Scroll` event. Nevertheless, the concept of a method is useful to describe functions that can only be invoked by a program and are not directly accessible to the user.

Object Management Functions

This is a set of system functions used to create objects, set properties, respond to events, and call methods.

1.2 Objects Categorised

Section 1.3

System Objects	
<u>Root</u>	system-level object
<u>Printer</u>	for hard-copy output
<u>Clipboard</u>	provides access to Windows clipboard
Container Objects	
<u>CoolBand</u>	represents a band in a CoolBar
<u>CoolBar</u>	a container for CoolBand objects
<u>Form</u>	top-level Window
<u>MDIClient</u>	container for MDI windows
<u>SubForm</u>	acts as an MDI window or a constrained Form
<u>Group</u>	a frame for grouping Buttons and other objects
<u>Static</u>	a frame for drawing and clipping graphics
<u>StatusBar</u>	ribbon status bar
<u>TabBar</u>	contains TabBtns (tabs)
<u>TabControl</u>	contains TabButtons (tabs)
<u>ToolBar</u>	ribbon tool bar
<u>ToolControl</u>	standard Windows tool control
<u>PropertySheet</u>	contains PropertyPages
<u>PropertyPage</u>	tabbed or paged container for other controls
<u>Splitter</u>	divides a container into panes
Menu	
<u>MenuBar</u>	pull-down menu bar
<u>Menu</u>	pop-up menu
<u>MenuItem</u>	selects an option or action
<u>Separator</u>	separator between items
Action	
<u>Button</u>	selects an option
<u>ToolButton</u>	performs an action or selects an option

System Objects	
<u>TabBtn</u>	selects a tabbed SubForm
<u>TabButton</u>	selects a tabbed SubForm
<u>Scroll</u>	scroll bar
<u>UpDown</u>	spin buttons
<u>Locator</u>	graphical (positional) input device
<u>Timer</u>	generates events at regular intervals
<u>SysTrayItem</u>	an item that you can create in the Windows System Tray
Information	
<u>Animation</u>	displays simple animations from .AVI files
<u>Label</u>	displays static text
<u>StatusField</u>	displays status information
<u>MsgBox</u>	displays a message box
<u>TipField</u>	displays pop-up context sensitive help
<u>ProgressBar</u>	displays the progress of a lengthy operation
Input & Selection	
<u>Calendar</u>	displays a month calendar control
<u>DateTimePicker</u>	an editable date/time field with an optional drop-down Calendar.
<u>ColorButton</u>	allows the user to select a colour.
<u>Grid</u>	displays a data matrix as a spreadsheet
<u>Edit</u>	text input field
<u>ButtonEdit</u>	text input field with dropdown button
<u>RichEdit</u>	text input with word-processing capabilities
<u>Spinner</u>	input field with spin buttons
<u>List</u>	for selecting an item
<u>ListView</u>	displays a collection of items for selection
<u>Combo</u>	edit field with selectable list of choices

System Objects	
<u>ComboEx</u>	extended version of <u>Combo</u>
<u>TreeView</u>	displays a hierarchical collection of items
<u>TrackBar</u>	a slider control for analogue input/output
<u>FileBox</u>	prompts user to select a file
<u>BrowseBox</u>	prompts user to browse for a folder or other resource.
Resource	
<u>Font</u>	loads a font
<u>Bitmap</u>	defines a bitmap
<u>Icon</u>	defines an icon
<u>ImageList</u>	defines a collection of bitmaps or icons
<u>Metafile</u>	loads a Windows Metafile
<u>Cursor</u>	defines a cursor
Graphical Output	
<u>Circle</u>	draws a circle
<u>Ellipse</u>	draws an ellipse
<u>Marker</u>	draws a series of polymarkers
<u>Poly</u>	draws lines
<u>Rect</u>	draws rectangles
<u>Image</u>	displays Bitmaps, Icons and Metafiles
<u>Text</u>	draws graphical text
Miscellaneous	
<u>ActiveXContainer</u>	represents the application hosting a Dyalog APL ActiveXControl
<u>ActiveXControl</u>	represents an ActiveX control written in Dyalog APL
<u>HTMLRenderer</u>	displays HTML content
<u>NetClient</u>	provides access to .NET Classes
<u>NetControl</u>	instantiates a .NET Control.
<u>NetType</u>	exports an APL namespace as a Net Class

System Objects	
<u>OCXClass</u>	provides access to OLE Custom Controls
<u>OLEClient</u>	provides access to OLE Automation objects
<u>OLEServer</u>	enables APL to act as an OLE Automation server
<u>SM</u>	specifies a window for □SM (character mode interface)
<u>TCPSocket</u>	provides an interface to TCP/IP sockets

1.3 Objects A-Z

Section 1.2

Object	Description
<u>ActiveXContainer</u>	represents the application hosting a Dyalog APL ActiveXControl
<u>ActiveXControl</u>	implements an ActiveX control written in Dyalog APL
<u>Animation</u>	plays simple AVIs
<u>Bitmap</u>	can be used to fill an area, or as a background pattern
<u>BrowseBox</u>	allows the user to browse for a folder or other resource
<u>Button</u>	used to perform a task or select an option
<u>ButtonEdit</u>	single-line input field with custom button
<u>Calendar</u>	displays a month calendar control
<u>Circle</u>	draws circles, arcs and pies
<u>Clipboard</u>	provides access to the Windows clipboard
<u>ColorButton</u>	allows the user to select a colour
<u>Combo</u>	combines text entry field with list of choices
<u>ComboEx</u>	an extended version of the Combo object that provides additional features including item images
<u>CoolBand</u>	represents a band in a CoolBar
<u>CoolBar</u>	acts as a container for CoolBand objects
<u>Cursor</u>	creates user-defined cursor
<u>DateTimePicker</u>	an editable date/time field with an optional drop-down Calendar
<u>Edit</u>	single or multi-line edit box for entering data
<u>Ellipse</u>	draws ellipses, elliptical arcs and pies
<u>FileBox</u>	standard File Selection dialog box
<u>Font</u>	a font resource
<u>Form</u>	window that acts as a container for other objects
<u>Grid</u>	spreadsheet object for editing data
<u>Group</u>	used to group a related set of controls together visually, and to impose "radio-button" behaviour
<u>HTMLRenderer</u>	displays HTML content

Object	Description
<u>Icon</u>	can be displayed, or used when a <u>Form</u> is minimized
<u>Image</u>	graphical object for displaying bitmaps and icons
<u>ImageList</u>	specifies a collection of bitmaps or icons for a <u>ListView</u> or <u>TreeView</u>
<u>Label</u>	fixed text that the user cannot change
<u>List</u>	list of items from which the user can choose
<u>ListView</u>	collection of items from which the user can choose
<u>Locator</u>	moving line, rectangle or ellipse for graphics input
<u>Marker</u>	draws marker at a series of points
<u>MDIClient</u>	provides Multiple Document Interface (MDI) behaviour
<u>Menu</u>	displays a pulldown or pop-up menu
<u>MenuBar</u>	displays list of pulldown menus across top of <u>Form</u>
<u>MenuItem</u>	component of a <u>Menu</u> that performs action or makes choice
<u>Metafile</u>	provides access to Windows <u>Metafile</u> s
<u>MsgBox</u>	displays message in dialog box and waits for response
<u>NetClient</u>	Provides access to .NET Classes
<u>NetControl</u>	Provides access to .NET controls.
<u>NetType</u>	Exports a namespace as a .NET class
<u>OCXClass</u>	provides access to OLE Custom Controls
<u>OLEClient</u>	provides access to OLE Automation objects
<u>OLEServer</u>	enables APL to act as an OLE Automation server
<u>Poly</u>	draws lines, polygons and filled areas
<u>Printer</u>	controls output to a printer
<u>ProgressBar</u>	indicates the progress of a lengthy operation
<u>PropertyPage</u>	tabbed or paged container for other controls
<u>PropertySheet</u>	container for PropertyPages
<u>Rect</u>	draws filled and unfilled rectangles

Object	Description
<u>RichEdit</u>	an edit box with word-processing capabilities
<u>Root</u>	the system object that is the progenitor of all others
<u>Scroll</u>	horizontal or vertical scrollbar
<u>Separator</u>	horizontal or vertical line in <u>Menu</u> , or vertical break in <u>MenuBar</u>
<u>SM</u>	allows <code>WSM</code> and <code>WSR</code> to be used with GUI objects
<u>Spinner</u>	input field with spin buttons
<u>Splitter</u>	divides a <u>Form</u> or <u>SubForm</u> into resizable panes
<u>Static</u>	frame or box used to contain graphics
<u>StatusBar</u>	manages a set of <u>StatusField</u> objects
<u>StatusField</u>	displays context-sensitive help, or keyboard status
<u>SubForm</u>	child <u>Form</u> that is constrained within its parent
<u>SysTrayItem</u>	represents an item that you can create in the Windows system tray
<u>TabBar</u>	manages a set of <u>TabBtn</u> objects
<u>TabBtn</u>	brings forward an associated <u>SubForm</u>
<u>TabButton</u>	represents a tab or button in a <u>TabControl</u>
<u>TabControl</u>	represents the standard Windows tab control
<u>TCPSocket</u>	provides an interface to TCP/IP sockets
<u>Text</u>	displays or prints arbitrary text
<u>Timer</u>	generates events at regular intervals
<u>TipField</u>	displays pop-up context-sensitive help
<u>ToolBar</u>	manages a block of controls including <u>Button</u> s
<u>ToolButton</u>	represents a or button in a <u>ToolControl</u>
<u>ToolControl</u>	represents the standard Windows tool control
<u>TrackBar</u>	slider control for analogue input/output
<u>TreeView</u>	displays a hierarchical list of items
<u>UpDown</u>	a pair of arrow buttons

1.4 Properties A-Z

Property	Description
<u>Accelerator</u>	specifies keystroke that will generate <u>Select</u> event
<u>Active</u>	determines whether object can generate events
<u>Align</u>	determines the position of text or for <u>Scroll</u> object
<u>AlignChar</u>	specifies a character for column alignment in a <u>Grid</u>
<u>AlwaysShowBorder</u>	specifies how the current cell in a <u>Grid</u> is displayed when the Grid loses focus
<u>AlwaysShowSelection</u>	specifies how the highlighted selection in an object is displayed when the object loses focus
<u>APLVersion</u>	reports the version of Dyalog being run
<u>ArcMode</u>	determines how arcs are drawn (<u>Ellipse</u>)
<u>AsChild</u>	determines if an <u>HTMLRenderer</u> object is displayed in a separate top-level window or as a child of another object.
<u>Attach</u>	specifies how object is reconfigured when parent is resized
<u>AutoArrange</u>	whether or not items in a <u>ListView</u> are rearranged automatically
<u>AutoBrowse</u>	specifies whether or not functions and variables are fixed when an <u>OLEClient</u> is created
<u>AutoConf</u>	how a child reacts to its parent being resized
<u>AutoExpand</u>	whether rows and columns are added to a <u>Grid</u>
<u>BandBorders</u>	Specifies whether or not narrow lines are drawn to separate adjacent bands in a <u>CoolBar</u>
<u>BCol</u>	specifies background colour
<u>Bits</u>	defines the pattern for a <u>Bitmap</u> , <u>Cursor</u> , or <u>Icon</u>
<u>Border</u>	determines whether or not an object has a border
<u>BtnPix</u>	associates <u>Bitmap</u> s with <u>Button</u> , <u>Menu</u> and <u>MenuItem</u> objects
<u>Btns</u>	determines the buttons shown in a <u>MsgBox</u>
<u>ButtonsAcceptFocus</u>	affects the keyboard handling in a <u>ToolControl</u>

Property	Description
<u>CalendarCols</u>	Specifies the colours used for various elements in the <u>Calendar</u> object
<u>Cancel</u>	used to associate the Esc key with a particular <u>Button</u>
<u>Caption</u>	specifies text in an object
<u>CBits</u>	represents the picture in a <u>Bitmap</u> object
<u>CEFVersion</u>	reports the version of the Chromium Embedded Framework (CEF) that is being used.
<u>CellFonts</u>	specifies fonts to be used by the cells in a <u>Grid</u>
<u>CellHeights</u>	specifies the heights of the cells in a <u>Grid</u>
<u>CellSelect</u>	specifies the type(s) of selection a user can perform in a <u>Grid</u>
<u>CellSet</u>	flags which cells in a <u>Grid</u> have values and which are empty
<u>CellTypes</u>	specifies the type of the cells in a <u>Grid</u>
<u>Cells</u>	specifies the widths of cells in a <u>Grid</u>
<u>Changed</u>	flags whether or not data in an object has been changed
<u>CharFormat</u>	specifies character formatting in a <u>RichEdit</u>
<u>CharSet</u>	specifies the character encoding of a <u>Font</u> object
<u>CheckBoxes</u>	specifies whether or not check boxes are displayed alongside items in a <u>ListView</u> or <u>TreeView</u> object
<u>Checked</u>	determines whether check mark is displayed by <u>MenuItem</u>
<u>ChildEdge</u>	specifies whether or not a <u>CoolBand</u> leaves space above and below its child window
<u>ChildList</u>	reports the types of object that may be children of an object
<u>CircleToday</u>	specifies whether or not a circle is drawn around the Today date in a <u>Calendar</u> object
<u>ClassID</u>	reports the CLSID of an <u>OLEClient</u> or <u>OLEServer</u>
<u>ClassName</u>	

Property	Description
	specifies the name of the OLE object to which an <u>OLEClient</u> object is to be connected
<u>ClipCells</u>	specifies whether or not columns of a <u>Grid</u> are clipped
<u>CMap</u>	defines a colour map for a <u>Bitmap</u> or <u>Icon</u>
<u>ColLineTypes</u>	specifies the appearance of vertical grid lines in a <u>Grid</u>
<u>ColSortImages</u>	specifies the images to be used to display sort images in the column titles of a <u>Grid</u>
<u>ColTitle3D</u>	specifies the appearance of column titles for a <u>ListView</u>
<u>ColTitleAlign</u>	specifies the alignment of column titles
<u>ColTitleBCol</u>	specifies the background colour for column titles
<u>ColTitleDepth</u>	specifies the structure of hierarchical column titles for a <u>Grid</u>
<u>ColTitleFCol</u>	specifies the colour of the row titles in a <u>Grid</u>
<u>ColTitles</u>	specifies the column titles for a <u>Grid</u> or <u>ListView</u>
<u>Column</u>	specifies the width of columns for a multi-column <u>List</u>
<u>Container</u>	the Object Representation of an <u>ActiveXContainer</u> object
<u>Coord</u>	specifies the coordinate system for an object
<u>CornerTitleBCol</u>	specifies the background colour of the corner rectangle in a <u>Grid</u>
<u>Cue</u>	specifies text displayed in an empty input field
<u>CurCell</u>	identifies the current cell in a <u>Grid</u>
<u>CurrentColor</u>	Specifies the currently selected colour in a <u>ColorButton</u> object
<u>CurrentState</u>	reports the current state of a <u>TCPSocket</u> object
<u>CursorObj</u>	associates a cursor with an object
<u>CustomColors</u>	Specifies the custom colors associated with a <u>ColorButton</u> object
<u>Decimals</u>	specifies the number of decimal places for a Numeric field

Property	Description
<u>Default</u>	nominates a <u>Button</u> to be selected with the Enter key
<u>DefaultColors</u>	Specifies the default colours associated with a <u>ColorButton</u> object
<u>Depth</u>	specifies the structure of items for a <u>TreeView</u>
<u>DevCaps</u>	reports the device capabilities of the screen or printer
<u>Directory</u>	specifies the directory for a <u>FileBox</u>
<u>Divider</u>	controls the presence or absence of a recessed line in a ToolControl object
<u>Dragable</u>	specifies whether user may drag an object with the mouse
<u>DragItems</u>	specifies whether user may drag/drop items in a <u>ListView</u>
<u>DrawMode</u>	provides direct control over the low-level drawing operation performed by graphical objects
<u>EdgeStyle</u>	specifies 3-dimensional appearance
<u>EditLabels</u>	specifies whether or not the user may edit the labels in a <u>ListView</u> or <u>TreeView</u>
<u>Elevated</u>	affects the appearance of a CommandLink <u>Button</u>
<u>Encoding</u>	specifies how character data are encoded or translated for a <u>TCPSocket</u>
<u>EnterReadOnlyCells</u>	specifies whether or not the user may visit read-only cells in a <u>Grid</u>
<u>Event</u>	associates an event with a callback function or <i>Dyalog APL Language: Dq</i> termination
<u>EventList</u>	reports names of events generated by an object
<u>ExportedFns</u>	specifies the functions to be exposed as methods by an <u>OLEServer</u> object
<u>ExportedVars</u>	specifies the variables to be exposed as properties by an <u>OLEServer</u> object
<u>FCol</u>	specifies foreground colour
<u>FieldType</u>	specifies formatting and validation

Property	Description
<u>FileMode</u>	specifies the mode (read or write) for a <u>FileBox</u> object
<u>File</u>	specifies a filename
<u>FillCol</u>	specifies fill colour
<u>Filters</u>	specifies file filters for a <u>FileBox</u>
<u>FirstDay</u>	specifies the day that is considered to be the first day of the week for a <u>Calendar</u> object
<u>Fixed</u>	specifies whether a font is fixed-width or proportional
<u>FixedOrder</u>	specifies whether or not the <u>CoolBar</u> displays <u>CoolBands</u> in the same order
<u>FlatSeparators</u>	specifies whether or not separators are drawn between buttons in a <u>TabControl</u> object
<u>FontList</u>	provides a list of available fonts
<u>FontObj</u>	specifies the font to be used
<u>Formats</u>	data formats currently available from the <u>Clipboard</u> object
<u>FormatString</u>	defines a $\square$ FMT specification to be used to format a numeric value
<u>FStyle</u>	specifies fill style
<u>FullRowSelect</u>	specifies whether or not the entire row is highlighted when an item in a <u>ListView</u> or a <u>TreeView</u> is selected
<u>GridBCol</u>	specifies the background colour for a <u>Grid</u>
<u>GridFCol</u>	specifies the colour of (all) grid lines for a <u>Grid</u>
<u>GridLineFCol</u>	specifies grid line colours for a <u>Grid</u>
<u>GridLine</u>	specifies grid line widths for a <u>Grid</u>
<u>GridLines</u>	specifies whether or not lines are displayed between items in a <u>ListView</u> object
<u>GripperMode</u>	specifies whether or not a <u>CoolBand</u> has a gripper bar
<u>HAlign</u>	specifies horizontal text alignment
<u>Handle</u>	Returns the window handle of an object

Property	Description
<u>HasApply</u>	Specifies whether or not a <u>PropertySheet</u> has an Apply button
<u>HasButtons</u>	Specifies whether or not a <u>TreeView</u> object displays buttons
<u>HasClearButton</u>	Specifies whether or not an <u>Edit</u> object displays an ✕ button which may be used to clear the contents of the field.
<u>HasHelp</u>	Specifies whether or not a Help button is displayed or active
<u>HasLines</u>	Specifies whether or not a <u>TreeView</u> object displays tree lines
<u>HasTicks</u>	Specifies whether or not tick marks are displayed in a <u>TrackBar</u>
<u>HasToday</u>	Specifies whether or not the Today date is displayed in the bottom left corner of a Calendar object
<u>Header</u>	specifies whether or not column titles are displayed in a <u>ListView</u>
<u>HeaderImageIndex</u>	Specifies the images for each column title in a <u>ListView</u>
<u>HeaderImageList</u>	Specifies the images for column titles in a <u>ListView</u>
<u>HelpButton</u>	specifies whether or not a question (?) button appears in the title bar of a <u>Form</u>
<u>HelpFile</u>	reports the name of a help file associated with an OLE Control
<u>HighlightHeaders</u>	Specifies whether or not row and column headings in a <u>Grid</u> are highlighted to indicate the current selection
<u>HintObj</u>	specifies the object in which to display the <u>Hint</u>
<u>Hint</u>	specifies the text for a context sensitive help message
<u>HotSpot</u>	specifies the hotspot for a <u>Cursor</u>
<u>HotTrack</u>	

Property	Description
	specifies whether or not the tabs or buttons in a <u>TabControl</u> object are automatically highlighted by the mouse pointer
<u>HScroll</u>	determines whether an object has a horizontal scrollbar
<u>HTML</u>	specifies the content to be rendered by an <u>HTMLRenderer</u> object.
<u>IconObj</u>	associates an <u>Icon</u> with a minimized <u>Form</u>
<u>ImageCount</u>	Reports the number of images in an <u>ImageList</u>
<u>ImageIndex</u>	Maps images in an <u>ImageList</u> to items in a <u>ListView</u> or <u>TreeView</u>
<u>ImageListObj</u>	Associates an <u>ImageList</u> with a <u>ListView</u> or <u>TreeView</u>
<u>Index</u>	position of items in <u>Combo</u> or <u>List</u> , or selected filter in a <u>FileBox</u>
<u>Input</u>	names of objects associated with cells of <u>Grid</u>
<u>InputMode</u>	specifies the behaviour of cursor keys etc. in a <u>Grid</u>
<u>InputModeKey</u>	specifies the key used to switch to Incell editing in a <u>Grid</u>
<u>InputProperties</u>	Specifies the mapping between the Values property of a <u>Grid</u> and properties of external objects (ActiveX controls and .NET classes) which are displayed in <u>Grid</u> cells
<u>InterceptedURLs</u>	specifies whether an <u>HTMLRenderer</u> will trigger an <u>HTTPRequest</u> event for the requested URL or will attempt to satisfy the request from the internet
<u>Interval</u>	frequency with which a <u>Timer</u> generates events
<u>Italic</u>	specifies whether or not a font is italic
<u>Items</u>	list of items
<u>ItemGroups</u>	specifies item groupings for a <u>ListView</u> object
<u>ItemGroupMetrics</u>	specifies caption, colours and spacing for grouped items in a <u>ListView</u> object
<u>Justify</u>	determines how text is justified within an object
<u>KeepBits</u>	determines how <u>Bitmap</u> objects are stored in the workspace

Property	Description
<u>KeepOnClose</u>	determines whether or not namespaces are retained when their GUI components are destroyed
<u>LastError</u>	provides information about the most recent error reported by OLE
<u>LateBind</u>	specifies whether or not Type Information is read when an <u>OLEClient</u> is instantiated, and when an <u>OLEServer</u> hosts an external COM object
<u>Limits</u>	specifies minimum and maximum values for an object
<u>LocalAddrName</u>	specifies the host name of your computer
<u>LocalAddr</u>	specifies the IP address of your computer
<u>Locale</u>	specifies the language in which the OLE server, attached to an <u>OLEClient</u> , exposes its methods and properties
<u>LocalPortName</u>	specifies the port name of the local service that you wish to offer as a TCP/IP server
<u>LocalPort</u>	identifies the <u>port number</u> associated with a <u>TCPSocket</u> object
<u>LStyle</u>	specifies line style
<u>LWidth</u>	specifies line width
<u>MapCols</u>	specifies whether button colours in bitmaps and icons in an <u>ImageList</u> are re-mapped to reflect the users colour preferences
<u>Masked</u>	specifies the type of image in an <u>ImageList</u>
<u>Mask</u>	specifies the mask for a <u>Cursor</u> or <u>Icon</u>
<u>MaxButton</u>	determines whether or not a <u>Form</u> has a "maximize" button
<u>MaxDate</u>	specifies the largest date that may be displayed by a <u>Calendar</u> object
<u>MaxLength</u>	specifies the maximum number of characters that the user may enter into an object
<u>MaxSelCount</u>	specifies the maximum number of contiguous days that the user may select in a <u>Calendar</u> object

Property	Description
<u>MDIActive</u>	specifies the name of the active <u>SubForm</u> in an MDI application
<u>MDIActiveObject</u>	specifies a ref to the active <u>SubForm</u> in an MDI application
<u>MDIMenu</u>	nominates a particular <u>Menu</u> to be the MDI application menu
<u>Metafile</u>	accesses clipboard data in Windows <u>Metafile</u> format
<u>MethodList</u>	reports the names of methods provided by an OLE Control
<u>MinButton</u>	determines whether a <u>Form</u> has a "minimize" button
<u>MinDate</u>	Specifies the smallest date that may be displayed by a <u>Calendar</u> object
<u>MonthDelta</u>	specifies the number of months by which a <u>Calendar</u> object scrolls when the user clicks its scroll buttons
<u>Moveable</u>	determines whether a <u>Form</u> may be moved on the screen
<u>MultiColumn</u>	specifies whether or not a <u>List</u> displays items in more than one column
<u>MultiLine</u>	determines whether or not the tabs or buttons will be arranged in multiple flights or multiple rows/columns in a <u>TabControl</u> or <u>ToolControl</u> object
<u>MultiSelect</u>	specifies whether or not the user can select more than one button in a <u>TabControl</u> at the same time
<u>NewLine</u>	specifies whether or not a <u>CoolBand</u> starts a new row in a <u>CoolBar</u>
<u>Note</u>	specifies additional text for a CommandLink <u>Button</u>
<u>OLEControls</u>	reports the names of OLE Controls installed on the computer
<u>OnTop</u>	specifies that a <u>Form</u> is permanently raised to the front or specifies the level at which a graphic is drawn on a <u>Grid</u>
<u>Orientation</u>	specifies the orientation of the <u>Printer</u>

Property	Description
<u>OtherButton</u>	Specifies whether or not a <u>ColorButton</u> object displays an <i>Other</i> button, that allows access to custom colours
<u>OverflowChar</u>	specifies the character to be displayed in a <u>Grid</u> cell to indicate that its contents will not fit
<u>Page</u>	specifies the width of the paper for a <u>RichEdit</u>
<u>PageActive</u>	specifies the name of the current <u>PropertyPage</u> in a <u>PropertySheet</u>
<u>PageActiveObject</u>	specifies a ref to the current <u>PropertyPage</u> in a <u>PropertySheet</u>
<u>PageSize</u>	specifies the size of the thumb in a scrollbar
<u>ParaFormat</u>	specifies paragraph formatting for a <u>RichEdit</u>
<u>Password</u>	specifies the symbol for a password field
<u>Picture</u>	specifies a Bitmap, <u>Icon</u> or <u>Metafile</u> object to be drawn
<u>PName</u>	specifies the device for a <u>Printer</u> object, or the face name for a <u>Font</u>
<u>Points</u>	specifies points for graphical objects
<u>Popup</u>	specifies the name of a (popup) <u>Menu</u> object that is associated with a <u>ToolButton</u>
<u>Posn</u>	specifies the position of an object within its parent
<u>PrintList</u>	reports the list of installed printers
<u>ProgressStyle</u>	specifies the appearance of a <u>ProgressBar</u> control
<u>PropList</u>	reports the list of properties applicable to the object
<u>QueueEvents</u>	specifies how events are handled for an instance of an OCXClass object (an ActiveX control)
<u>Radius</u>	specifies the radius for a <u>Circle</u>
<u>Range</u>	specifies the range of a scrollbar
<u>ReadOnly</u>	specifies whether or not the user may change data in an <u>Edit</u> or <u>Spinner</u> , or the <u>State</u> of a <u>Button</u>
<u>RealSize</u>	specifies the size of a placeable <u>Metafile</u>
<u>Redraw</u>	

Property	Description
	specifies whether or not an object is automatically redrawn when required
<u>RemoteAddrName</u>	specifies the host name of the remote computer to which you wish to make a TCP/IP connection
<u>RemoteAddr</u>	identifies the <u>port number</u> associated with a service on a remote computer
<u>RemotePortName</u>	specifies the port name of the remote service to which you wish to make a TCP/IP connection
<u>RemotePort</u>	identifies the <u>port number</u> associated with a service on a remote computer
<u>ReportBCol</u>	specifies background colours for items in a <u>ListView</u> in Report View
<u>ReportImageIndex</u>	specifies images for individual column titles in a <u>ListView</u> in Report View
<u>ReportInfo</u>	specifies additional information for a <u>ListView</u>
<u>ResizeCols</u>	specifies whether or not the user may alter the width of columns in a <u>Grid</u> by dragging
<u>ResizeColTitles</u>	specifies whether or not the user may alter the height of the column titles in a <u>Grid</u> by dragging
<u>ResizeRows</u>	specifies whether or not the user may alter the height of rows in a <u>Grid</u> by dragging
<u>ResizeRowTitles</u>	specifies whether or not the user may alter the width of the row titles in a <u>Grid</u> by dragging
<u>Rotate</u>	specifies the angle of rotation for a font
<u>RowHiddenDepth</u>	identifies which rows in a <u>Grid</u> are hidden
<u>RowLineTypes</u>	specifies the appearance of horizontal grid lines in a <u>Grid</u>
<u>Rows</u>	number of rows displayed in drop-down list of a <u>Combo</u>
<u>RowTitleAlign</u>	specifies the alignment of the row titles in a <u>Grid</u>
<u>RowTitleBCol</u>	specifies the background colour for row titles
<u>RowTitleDepth</u>	specifies the structure of hierarchical row titles in a <u>Grid</u>
<u>RowTitleFCol</u>	specifies the colour of the row titles in a <u>Grid</u>

Property	Description
<u>RowTitles</u>	specifies the row titles for a <u>Grid</u>
<u>RowTreeDepth</u>	specifies the structure of rows for a <u>Grid</u>
<u>RowTreeImages</u>	specifies the images (bitmaps) for the treeview items in a <u>Grid</u>
<u>RowTreeStyle</u>	specifies the appearance of treeview items in a <u>Grid</u>
<u>RTFText</u>	specifies the contents of an object in Rich Text Format
<u>RunMode</u>	specifies the way in which an <u>OLEServer</u> object serves multiple clients
<u>ScrollOpposite</u>	specifies that unneeded tabs scroll to the opposite side of a <u>TabControl</u>
<u>SelDate</u>	identifies the range of dates that is currently selected in a <u>Calendar</u> object
<u>SelectionBorderWidth</u>	specifies the width of the border drawn around the block of selected cells in a <u>Grid</u>
<u>SelectionColor</u>	specifies the colour of the block of selected cells in a <u>Grid</u>
<u>SelectionColorAlpha</u>	specifies the transparency of the block of selected cells in a <u>Grid</u>
<u>SellImageIndex</u>	specifies which images in an <u>ImageList</u> are used to identify selected items in a <u>TreeView</u>
<u>SellItems</u>	specifies the selected item(s) in an object
<u>SelRange</u>	specifies the selected range in a <u>TrackBar</u>
<u>SelText</u>	specifies the selected text in an <u>Edit</u> or <u>Combo</u>
<u>ServerVersion</u>	specifies the version number of an <u>OLEServer</u> object
<u>ShowCaptions</u>	specifies whether or not the captions of individual <u>ToolButton</u> objects are drawn
<u>ShowCueWhenFocused</u>	specifies whether or not a Cue is displayed when the input field receives the input focus
<u>ShowDropDown</u>	specifies whether or not a drop-down menu symbol is drawn in a <u>ColorButton</u> or alongside <u>ToolButton</u> objects
<u>ShowInput</u>	

Property	Description
	determines how <u>Combo</u> and <u>Button</u> objects associated with cells of a <u>Grid</u> are displayed
<u>ShowSession</u>	specifies whether or not the APL Session window is displayed when an <u>OLEServer</u> object is started by an OLE client
<u>ShowThumb</u>	specifies whether or not the thumb is displayed in a <u>TrackBar</u>
<u>SingleClickExpand</u>	specifies whether or not an item in a <u>TreeView</u> control is expanded when the user selects the item
<u>Sizeable</u>	specifies whether user may resize an object using the mouse
<u>Size</u>	specifies the size of an object
<u>SocketNumber</u>	reports the Window handle of the socket attached to the <u>TCPSocket</u> object
<u>SocketType</u>	specifies the type of the TCP/IP socket for a <u>TCPSocket</u> object
<u>SortItems</u>	specifies whether or not the items in a <u>List</u> object are sorted
<u>SplitObj1</u>	specifies the name of an object managed by a <u>Splitter</u>
<u>SplitObj2</u>	specifies the name of an object managed by a <u>Splitter</u>
<u>Start</u>	specifies start angles for arcs of <u>Circle</u> and <u>Ellipse</u> objects
<u>State</u>	specifies the state of a <u>Button</u> or <u>Form</u>
<u>Step</u>	increments for movement
<u>Style</u>	specifies the style of an object
<u>SysMenu</u>	determines whether a <u>Form</u> has a standard system menu
<u>TabFocus</u>	specifies the focus behaviour for the <u>TabControl</u> object
<u>TabIndex</u>	Specifies the tabbing order for controls
<u>TabJustify</u>	Specifies the positions at which the picture and caption are drawn within a TabButton
<u>TabObj</u>	specifies name of <u>SubForm</u> associated with a <u>TabBtn</u>

Property	Description
<u>TabSize</u>	specifies the size of fixed size tabs or buttons in a <u>TabControl</u> object
<u>TargetState</u>	reports the intended final state of a <u>TCPSocket</u> object
<u>Text</u>	specifies/reports the text in an <u>Edit</u> , <u>MsgBox</u> , or <u>Combo</u>
<u>TextSize</u>	reports the bounding rectangle for text
<u>Thumb</u>	specifies the position of the thumb in a scrollbar
<u>ThumbRect</u>	reports the extents of the thumb in a <u>TrackBar</u>
<u>TickAlign</u>	specifies the alignment of tick marks in a <u>TrackBar</u>
<u>TickSpacing</u>	specifies the spacing of tick marks in a <u>TrackBar</u>
<u>TipObj</u>	specifies the object in which to display the <u>Tip</u>
<u>Tip</u>	specifies the text for a pop-up help message
<u>TitleHeight</u>	specifies the height of the column titles in a <u>Grid</u>
<u>Title</u>	specifies the width of the row titles in a <u>Grid</u>
<u>Today</u>	specifies today's date in a <u>Calendar</u> object
<u>ToolboxBitmap</u>	reports the bitmap image associated with an OLE Control
<u>TrackRect</u>	reports the extents of the slider in a <u>TrackBar</u>
<u>Translate</u>	specifies whether or not an object performs ☐ AV translation
<u>Transparent</u>	specifies whether or not a <u>ToolControl</u> is transparent
<u>TypeLibFile</u>	reports the name of the file in which the Type Library for an <u>OLEServer</u> object is stored
<u>TypeLibID</u>	reports the value of the globally unique identifier (<u>GUID</u>) of the Type Library associated with an <u>OLEServer</u> object
<u>TypeList</u>	reports the names of data types associated with an OLE Control
<u>Type</u>	specifies the type of an object
<u>Underline</u>	specifies whether or not a font is underlined

Property	Description
<u>URL</u>	specifies the "root" url of the server component of the <u>HTMLRenderer</u> object for internal processing, or it specifies an external URL to be retrieved via the internet
<u>ValidIfEmpty</u>	specifies whether or not an empty <u>Edit</u> object is deemed to be valid
<u>VAlign</u>	specifies vertical text alignment
<u>Value</u>	value of a number, date or time in an <u>Edit</u> or <u>Label</u> object
<u>Values</u>	the data matrix in a <u>Grid</u>
<u>VariableHeight</u>	specifies whether or not a <u>CoolBar</u> displays bands at the minimum required height, or all the same height
<u>View</u>	specifies how items in a <u>ListView</u> are arranged
<u>Visible</u>	specifies whether or not an object is currently visible
<u>VScroll</u>	specifies whether or not an object has a vertical scrollbar
<u>WantsReturn</u>	specifies how the Enter key is treated by an edit box
<u>WeekNumbers</u>	specifies whether or not a <u>Calendar</u> object displays week numbers
<u>Weight</u>	specifies the weight (boldness) of a font
<u>WordFormat</u>	specifies word formatting for a <u>RichEdit</u>
<u>Wrap</u>	specifies whether or not an object wraps its contents
<u>XRange</u>	specifies origin and scale on the x-axis
<u>Yield</u>	specifies how frequently Dyalog APL/W yields control
<u>YRange</u>	specifies origin and scale on the y-axis

1.5 Events A-Z

Event	Num	Description
All	0	This is shorthand for "all events". It is not an event in its own right. See <i><u>Special Case for All events</u></i> .
<u>ActivateApp</u>	139	User has switched to/from the APL application
<u>AddCol</u>	153	appended a column to a <u>Grid</u> object
<u>AddRow</u>	152	appended a row to a <u>Grid</u> object
<u>AmbientChanged</u>	533	reported when any of the ambient properties change in an application hosting an <u>ActiveXControl</u> object
<u>AnimStarted</u>	294	reported just before an AVI clip starts playing in an <u>Animation</u> object.
<u>AnimStopped</u>	295	reported just after an AVI clip stops playing in an <u>Animation</u> object.
<u>BadValue</u>	180	attempted to leave <u>Edit</u> object with invalid text
<u>BalloonHide</u>	862	reported when a BalloonTip is displayed
<u>BalloonShow</u>	861	reported when a BalloonTip disappears
<u>BalloonTimeout</u>	863	reported when a BalloonTip is dismissed
<u>BalloonUserClick</u>	864	reported when a BalloonTip is clicked
<u>BeginEditLabel</u>	300	signals the start of an edit operation in a <u>ListView</u> or <u>TreeView</u> object
<u>CalendarDbClick</u>	273	reported when the user double-clicks the left mouse button over a <u>Calendar</u> object
<u>CalendarDown</u>	271	reported when the user depresses the left mouse button over a <u>Calendar</u> object
<u>CalendarMove</u>	274	reported when the user moves the left mouse button over a <u>Calendar</u> object
<u>CalendarUp</u>	272	reported when the user releases the left mouse button over a <u>Calendar</u> object
<u>CellChanged</u>	164	modified and updated the contents of a cell in a <u>Grid</u> object
<u>CellChange</u>	150	modified the contents of a cell in a <u>Grid</u> object

Event	Num	Description
<u>CellDbClick</u>	163	double-clicked the mouse on a cell in a <u>Grid</u>
<u>CellDown</u>	161	depressed a mouse button over a cell in a <u>Grid</u>
<u>CellError</u>	157	input invalid data into a cell in a <u>Grid</u>
<u>CellMove</u>	151	moved to a new cell in a <u>Grid</u> object
<u>CellOver</u>	160	mouse moved over a cell in a <u>Grid</u> object
<u>CellUp</u>	162	released a mouse button over a cell in a <u>Grid</u>
<u>Change</u>	36	altered the text in an <u>Edit</u> or <u>Combo</u>
<u>ClickComment</u>	225	user has clicked the mouse in a comment window of a <u>Grid</u>
<u>ClipChange</u>	120	data in the clipboard has changed
<u>Close</u>	33	<u>Form</u> is about to be closed
<u>CloseUp</u>	46	reported by a <u>DateTimePicker</u> object just before the drop-down calendar is hidden
<u>ColorChange</u>	430	user has selected a colour in a <u>ColorButton</u>
<u>ColumnClick</u>	320	user has clicked on a heading in a <u>ListView</u>
<u>Configure</u>	31	position and/or size of an object is about to change
<u>ContextMenu</u>	410	user has performed a standard Windows action to display a ContextMenu
<u>Create</u>	34	reported after an object has been created
<u>DateTimeChange</u>	267	user has changed the <u>DateTime</u> value in a <u>DateTimePicker</u> object
<u>DDE</u>	50	DDE message has been received or sent
<u>DisplayChange</u>	137	user has changed screen resolution or number of colours
<u>DockAccept</u>	483	reported by a host object just before it accepts a client object docking operation
<u>DockCancel</u>	485	user has aborted a docking operation by pressing Escape
<u>DockEnd</u>	484	

Event	Num	Description
		reported by a client object after it has been successfully docked in a host object
<u>DockMove</u>	481	reported by a host object when a dockable object (the client) is dragged over it
<u>DockRequest</u>	482	reported by a client object just before it is docked in a host object, when the user releases the mouse button
<u>DockStart</u>	480	reported by a client object when the user starts to drag it using the mouse
<u>DoPopup</u>	846	triggered when the <u>HTMLRenderer</u> client attempts to open a new window.
<u>DragDrop</u>	11	moved an object using drag & drop operation
<u>DropDown</u>	45	user has clicked the drop-down button in an object, just before the drop-down part is displayed
<u>DropFiles</u>	450	user has drag/dropped a file icon onto an object
<u>DropObjects</u>	455	user has drag/dropped an object icon from the Workspace Explorer or Find Objects tool onto an object
<u>DyalogCustomMessage1</u>	95	allows external applications and dynamic link libraries to insert events into the Dyalog APL/W message queue
<u>EndEditLabel</u>	301	signals completion of an edit operation in a <u>ListView</u> or <u>TreeView</u> object
<u>EndSplit</u>	282	reported when user releases the left mouse button to signify the end of a drag operation on a <u>Splitter</u> object
<u>ExitApp</u>	132	selected End Task from the Windows Task List
<u>ExitWindows</u>	131	requested Windows to terminate
<u>Expanding</u>	302	reported by a <u>TreeView</u> object as it is about to expand
<u>Expose</u>	32	exposed part or all of a <u>Form</u> or a <u>Static</u>

Event	Num	Description
<u>FileBoxCancel</u>	72	selected the <u>Cancel</u> button in a <u>FileBox</u>
<u>FileBoxOK</u>	71	selected the Ok button in a <u>FileBox</u>
<u>FontCancel</u>	242	user has pressed the <u>Cancel</u> button or closed the font selection dialog box
<u>FontOK</u>	241	user has pressed the OK button in the font selection dialog box
<u>FrameContextMenu</u>	411	user has clicked the right mouse button over the non-client area of an object
<u>GesturePan</u>	494	generated by a Pan gesture
<u>GesturePressAndTap</u>	497	generated by a press-and-tap gesture
<u>GestureRotate</u>	495	generated by a rotate gesture
<u>GestureTwoFingerTap</u>	496	generated by a two-finger-tap gesture
<u>GestureZoom</u>	493	generated by a zoom gesture
<u>GetDayStates</u>	266	reported when a <u>Calendar</u> object requires the Dyalog program to provide <i>day state</i> information for the range of dates it is about to display
<u>GetTipText</u>	325	reported by a <u>TreeView</u> or <u>ListView</u> object just before it displays a tip for a specific row
<u>GotFocus</u>	40	user has moved the keyboard focus to a new object
<u>GridCopy</u>	191	reported when the user presses Ctrl+Insert and there are selected cells in the <u>Grid</u>
<u>GridCopyError</u>	196	the user has pressed Ctrl+Insert and there is more than one block of selected cells in the <u>Grid</u> and the blocks are non-conformable
<u>GridCut</u>	190	the user has pressed Shift+Delete and there are selected cells in the <u>Grid</u>
<u>GridDelete</u>	193	the user has pressed Delete and there are selected cells in the <u>Grid</u>
<u>GridDropSel</u>	195	the user has drag/dropped a selected block of cells in the <u>Grid</u>

Event	Num	Description
<u>GridKeyPress</u>	24	user has pressed a key in a <u>Grid</u> cell
<u>GridPaste</u>	192	the user has pressed Shift+Insert and there is data in the clipboard that is in a suitable format for the <u>Grid</u>
<u>GridPasteError</u>	194	the user has pressed Shift+Insert and there is data in the clipboard, but the system is unable to paste the data into the <u>Grid</u>
<u>GridSelect</u>	165	user has changed the selection in a <u>Grid</u>
<u>Help</u>	400	user has clicked the Question (?) button and then clicked on an object
<u>HideComment</u>	224	generated just before a comment window is hidden as a result of the user moving the mouse-pointer away from a commented cell
<u>HScroll</u>	39	requested movement in horizontal scrollbar of <u>Form</u>
<u>HThumbDrag</u>	442	user has dragged the thumb of the horizontal scrollbar of a <u>Form</u> or <u>SubForm</u>
<u>HttpRequest</u>		raised when an <u>HTMLRenderer</u> issues a request for a url on the internal server
<u>Idle</u>	130	generated when system is idle
<u>IndexChanged</u>	210	user has scrolled the data in a <u>Grid</u> so that the value of its <u>Index</u> property has changed
<u>ItemDbClick</u>	342	user has double-clicked on an item in a <u>ListView</u> or <u>TreeView</u> object
<u>ItemDown</u>	340	user has depressed the left mouse button over an item in a <u>ListView</u> or <u>TreeView</u> object
<u>ItemUp</u>	341	user has released the left mouse button over an item in a <u>ListView</u> or <u>TreeView</u> object
<u>KeyError</u>	23	pressed an invalid key on the keyboard
<u>KeyPress</u>	22	pressed a key on the keyboard
<u>Locator</u>	80	terminated interaction with a <u>Locator</u> object
<u>LostFocus</u>	41	object has lost the input focus

Event	Num	Description
<u>MDIActivate</u>	42	MDI <u>SubForm</u> becomes the active one
<u>MDIDeactivate</u>	43	MDI <u>SubForm</u> is deactivated
<u>MouseDbClick</u>	5	double-clicked a mouse button
<u>MouseDown</u>	1	pressed mouse button down
<u>MouseEnter</u>	6	moved the mouse into the object
<u>MouseLeave</u>	7	moved the mouse out of the object
<u>MouseMove</u>	3	user has moved the mouse
<u>MouseUp</u>	2	user released a mouse button
<u>MouseWheel</u>	8	user rotated the mouse wheel
<u>MsgBtn1</u>	61	user selected first button in a <u>MsgBox</u>
<u>MsgBtn2</u>	62	user selected second button in a <u>MsgBox</u>
<u>MsgBtn3</u>	63	user selected third button in a <u>MsgBox</u>
<u>PageActivate</u>	360	user has switched to a new <u>PropertyPage</u>
<u>PageApply</u>	350	user has clicked Apply in a <u>PropertySheet</u>
<u>PageBack</u>	353	user has clicked Back in a <u>PropertySheet</u>
<u>PageCancel</u>	351	user has clicked Cancel in a <u>PropertySheet</u>
<u>PageChanged</u>	356	user has altered <u>Changed</u> property of a <u>PropertyPage</u>
<u>PageDeactivate</u>	361	user has switched to a new <u>PropertyPage</u>
<u>PageFinish</u>	355	user has clicked Finish in a <u>PropertySheet</u>
<u>PageHelp</u>	352	user has clicked Help in a <u>PropertySheet</u>
<u>PageNext</u>	354	user has clicked Next in a <u>PropertySheet</u>
<u>PreCreate</u>	534	Reported when an instance of an ActiveXControl is created
<u>Protected</u>	470	user has attempted to alter protected text in a <u>RichEdit</u> object
<u>Retracting</u>	304	reported by a <u>TreeView</u> object as it is about to retract

Event	Num	Description
<u>Scroll</u>	37	requested a movement of the thumb in scrollbar
<u>SelDateChange</u>	265	reported when the user changes the date that is selected in a <u>Calendar</u> object
<u>Select</u>	30	user has selected the object
<u>SelectCertificate</u>	848	triggered when an <u>HTMLRenderer</u> requests a resource that requires a certificate
<u>ServiceNotification</u>	94	reported when Windows requests a change in the state of an APL service
<u>SetColSize</u>	176	generated when column width changes in a <u>Grid</u>
<u>SetItemPosition</u>	322	generated when the user drag-drops an item in a <u>ListView</u>
<u>SetRowSize</u>	175	generated when row height changes in a <u>Grid</u>
<u>SetSpinnerText</u>	421	user has clicked a spin button in a <u>Spinner</u> object
<u>SetWizard</u>	365	user has clicked Next or Back in a <u>PropertySheet</u>
<u>ShowComment</u>	223	generated when the user rests the mouse pointer over a commented cell
<u>Spin</u>	420	user has clicked a spin button in a <u>Spinner</u> object
<u>Splitting</u>	281	reported while a <u>Splitter</u> object is being dragged, between a <u>StartSplit</u> and an <u>EndSplit</u>
<u>StartSplit</u>	280	reported when the user depresses the left mouse button over a <u>Splitter</u> object
<u>StateChange</u>	35	<u>Form</u> is about to change state
<u>SysColorChange</u>	134	system colour scheme has changed
<u>TCPAccept</u>	371	reported when a client connects to a server <u>TCPSocket</u> object
<u>TCPClose</u>	374	reported when the remote end of a TCP/IP connection breaks the connection
<u>TCPConnect</u>	372	reported when a server accepts the connection of a client <u>TCPSocket</u> object and is reported by the client
<u>TCPError</u>	370	

Event	Num	Description
		generated when a fatal TCP/IP error occurs and is reported by a <u>TCPSocket</u> object
<u>TCPGotAddr</u>	377	reported when a host name (specified by the <u>RemoteAddrName</u> or <u>LocalAddrName</u> property) is resolved to an IP address
<u>TCPGotPort</u>	378	reported when a port name (specified by the <u>RemotePortName</u> or <u>LocalPortName</u> property) is resolved to a port number
<u>TCPReady</u>	379	reported when the TCP/IP buffers are free and there is no data waiting to be sent in the internal APL queue
<u>TCPRecv</u>	373	reported when data is received by a <u>TCPSocket</u> object
<u>ThumbDrag</u>	440	user is dragging the thumb in a <u>TrackBar</u>
<u>Timer</u>	140	event generated by a <u>Timer</u> object
<u>VScroll</u>	38	requested movement in vertical scrollbar of <u>Form</u>
<u>VThumbDrag</u>	441	user has dragged the thumb of the vertical scrollbar of a <u>Form</u> or <u>SubForm</u>
<u>WebSocketClose</u>	843	triggered when the <u>HTMLRendered</u> client closes a WebSocket
<u>WebSocketError</u>	844	triggered when an error occurs on a WebSocket
<u>WebSocketReceive</u>	842	triggered when data is received over a WebSocket
<u>WebSocketUpgrade</u>	841	reported when the client component of an <u>HTMLRendered</u> object opens a WebSocket
<u>WinIniChange</u>	133	WIN.INI has changed

1.6 Methods A-Z

Method	Num	Description
<u>Abort</u>	103	abort a print job
<u>AddChildren</u>	310	adds child items to an item in a <u>TreeView</u>
<u>AddComment</u>	220	adds a comment to a <u>Grid</u>
<u>AddItems</u>	308	adds items to a <u>TreeView</u>
<u>Animate</u>	29	produces special effects when showing or hiding objects
<u>AnimClose</u>	291	closes the AVI file that is currently loaded in an <u>Animation</u> object.
<u>AnimOpen</u>	290	opens an AVI file in an <u>Animation</u> object.
<u>AnimPlay</u>	292	plays an AVI clip in an in an <u>Animation</u> object.
<u>AnimStop</u>	293	stops an AVI clip that is playing in an in an <u>Animation</u> object.
<u>Browse</u>	585	retained for compatibility with previous versions of Dyalog, but is no longer relevant and has no effect
<u>CancelToClose</u>	367	changes the buttons in a <u>PropertySheet</u>
<u>CellFromPoint</u>	200	converts from <u>Grid</u> coordinates to cell coordinates
<u>ChooseFont</u>	240	displays the standard Windows font selection box
<u>ColChange</u>	159	sets new values for a column of cells in a <u>Grid</u>
<u>ColSorted</u>	174	sets the sort image to be displayed in the column title of a <u>Grid</u>
<u>DateToIDN</u>	264	Converts a date from $\square$ TS format into an IDN suitable for use in a <u>Calendar</u> object
<u>DelCol</u>	155	deletes a column from a <u>Grid</u> object
<u>DelComment</u>	221	deletes a comment from a <u>Grid</u>
<u>DeleteChildren</u>	311	deletes child items from a parent item in a <u>TreeView</u>
<u>DeleteItems</u>	309	deletes items from a <u>TreeView</u>
<u>DeleteTypeLib</u>	521	removes a loaded Type Library from the workspace
<u>DelRow</u>	154	deletes a row from a <u>Grid</u> object

Method	Num	Description
<u>Detach</u>	270	detaches the GUI component from an object
<u>DuplicateColumn</u>	178	Duplicates a column in a <u>Grid</u> object
<u>DuplicateRow</u>	177	Duplicates a row in a <u>Grid</u> object
<u>ExecuteJavaScript</u>	839	Executes JavaScript in an <u>HTMLRenderer</u> object.
<u>FileRead</u>	90	causes a graphical object to be read from a file
<u>FileWrite</u>	91	causes a graphical object to be written to a file
<u>Flush</u>	135	flushes all pending output
<u>GetBuildID</u>	992	obtains the Build ID of a Dyalog executable.
<u>GetCellRect</u>	201	obtains the extents of a particular <u>Grid</u> cell
<u>GetCommandLine</u>	145	obtains the entire command line that was used to start Dyalog APL
<u>GetCommandLineArgs</u>	148	obtains the command line and arguments (as a nested vector) that was used to start Dyalog APL
<u>GetComment</u>	222	obtains the comment associated with a cell in a <u>Grid</u>
<u>GetEnvironment</u>	510	obtains information about one or more Dyalog parameters
<u>GetEventInfo</u>	551	obtains information about an OLE Control event
<u>GetFocus</u>	511	obtains the name of the object that currently has the input focus
<u>GetItemHandle</u>	313	obtains the window handle of an item in a <u>TreeView</u>
<u>GetItemPosition</u>	323	obtains the position of an item in a <u>ListView</u>
<u>GetItemState</u>	306	obtains the status of an item in a <u>TreeView</u>
<u>GetMethodInfo</u>	552	obtains information about an OLE Control method
<u>GetMinSize</u>	275	Obtains the minimum size that you must specify for a <u>Calendar</u> object for it to display a complete month
<u>GetParentItem</u>	312	obtains the index of the parent of an item in a <u>TreeView</u>

Method	Num	Description
<u>GetPropertyInfo</u>	550	obtains information about an OLE Control property
<u>GetServiceState</u>	92	obtains the state of an APL service
<u>GetTextSize</u>	146	obtains the size of the bounding rectangle of a text item in a given font
<u>GetTypeInfo</u>	553	obtains information about an OLE Control data type
<u>GetVisibleRange</u>	262	Obtains the range of dates that is currently visible in a <u>Calendar</u> object
<u>GreetBitmap</u>	138	Displays or remove a bitmap, typically during initialisation of a Dyalog runtime application
<u>IDNToDate</u>	263	Used to convert a date from an IDN into $\square$ TS format
<u>ListTypeLibs</u>	520	Obtains the names and CLSIDs of all the loaded Type Libraries.
<u>LockColumns</u>	227	Locks/unlocks columns in a <u>Grid</u> object
<u>LockRows</u>	226	Locks/unlocks rows in a <u>Grid</u> object
<u>MakeGIF</u>	261	Generates a GIF representation of a picture from a <u>Bitmap</u> object
<u>MakePNG</u>	260	Generates a PNG representation of a picture from a <u>Bitmap</u> object
<u>MDIArrange</u>	112	<u>MDIClient</u> to arrange minimised <u>SubForm</u> icons
<u>MDICascade</u>	110	<u>MDIClient</u> to overlap its <u>SubForm</u> s
<u>MDITile</u>	111	<u>MDIClient</u> to rearrange <u>SubForm</u> s in row or column
<u>NameFromHandle</u>	136	obtains the name of an object from its handle
<u>NewPage</u>	102	throws a new page on a <u>Printer</u>
<u>OLEAddEventSink</u>	540	Connects a named event sink to a COM object
<u>OLEDeleteEventSink</u>	541	Disconnects a named event sink from a COM object
<u>OLEListEventSinks</u>	542	Returns the names of event sinks that are currently connected to a COM object

Method	Num	Description
<u>OLEQueryInterface</u>	543	Used to obtain the methods and properties associated with a particular interface that is provided by a COM object
<u>OLERegister</u>	530	used to register an <u>OLEServer</u> object and may be used to install Dyalog OLE Servers as part of a run-time installation
<u>OLEUnregister</u>	531	used to unregister an <u>OLEServer</u> object that has previously been saved by Dyalog APL
<u>Print</u>	100	spools <u>Printer</u> output
<u>PrintToPDF</u>	845	writes the content displayed in an <u>HTMLRenderer</u> object to a specified file in Portable Document Format (pdf).
<u>ProgressStep</u>	250	increments the thumb in a <u>ProgressBar</u>
<u>RowChange</u>	158	sets new values for a row of cells in a <u>Grid</u>
<u>RowSetVisibleDepth</u>	173	set the maximum visible depth of data in rows of a <u>Grid</u>
<u>RTFPrint</u>	461	prints the contents of a <u>RichEdit</u>
<u>RTFPrintSetup</u>	460	displays a print setup dialog box
<u>SetCellSet</u>	171	changes <u>CellSet</u> property for a cell in a <u>Grid</u> .
<u>SetCellType</u>	156	changes <u>CellTypes</u> property for a cell in a <u>Grid</u>
<u>SetEventInfo</u>	547	Used to register an event that may be generated by an <u>ActiveXControl</u> object
<u>SetFinishText</u>	366	sets the caption of the Finish button in a Wizard-style <u>PropertySheet</u> .
<u>SetFnInfo</u>	545	Used to describe an APL function that is to be exported as a method, or as a property, of an <u>ActiveXControl</u> object
<u>SetServiceState</u>	93	sets the state of an APL service
<u>SetItemImage</u>	315	allocates a picture icon to an item in a <u>TreeView</u>
<u>SetItemState</u>	307	sets the status of an item in a <u>TreeView</u>
<u>SetMethodInfo</u>	556	

Method	Num	Description
		Used to describe a method that is exported by a COM object
<u>SetPropertyInfo</u>	554	Used to describe a property that is exported by a COM object
<u>Setup</u>	101	displays printer setup dialog box
<u>SetVarInfo</u>	546	Used to describe an APL variable that is to be exported as a property of an <u>ActiveXControl</u> object
<u>ShowBalloonTip</u>	860	displays a BalloonTip
<u>ShowDevTools</u>		shows or hides the CEF development tools window that displays information about <u>HTMLRenderer</u> content
<u>ShowHelp</u>	580	displays the help file associated with an OLE Control
<u>ShowItem</u>	316	displays a particular item in a <u>TreeView</u>
<u>ShowProperties</u>	560	displays the property sheet associated with an OLE Control
<u>TCPGetHostID</u>	376	obtains the IP Address of your PC
<u>TCPSend</u>	375	used to send data to a remote process connected to a <u>TCPSocket</u> object
<u>TCPSendPicture</u>	380	Transmits a picture represented by a <u>Bitmap</u> object to a TCP/IP socket
<u>Undo</u>	170	reverses last change made to a <u>Grid</u> object
<u>Wait</u>	147	same as executing □DQ on the object
<u>WebSocketSend</u>	847	sends data to a WebSocket

2 Objects

2.1 ActiveXContainer Object

Purpose: The ActiveXContainer object represents the application that is currently hosting an instance of an ActiveXControl object.

Parents ActiveXControl

Properties Type Event FontObj FCol BCol Data KeepOnClose MethodList ChildList EventList PropList

Methods Detach OLEQueryInterface

Events Close Create AmbientChanged

Description

An ActiveXContainer is used to represent the host application that is hosting an *ActiveXControl* object, and provides access to its ambient properties such as font, and colour.

An ActiveXContainer object is created using the *Container* property of the *ActiveXControl* object.

For example, the following expression, executed within an *ActiveXControl* instance creates an ActiveXContainer named 'CONT'

```
'CONT' = NS.WG.Container'
```

The ambient properties of the host application are reported by the *FontObj*, *BCol* and *FCol* properties which are all read-only.

The ActiveXContainer object supports the *AmbientChanged* event which is reported when any of the ambient properties change. This event allows the ActiveXContainer to react to such changes.

2.2 ActiveXControl Object

Purpose: The ActiveXControl object represents a Dyalog APL namespace as an ActiveX control.

Parents Form

Children ActiveXContainer Animation Bitmap BrowseBox Button ButtonEdit Calendar Circle Clipboard ColorButton Combo ComboEx CoolBar Cursor DateTimePicker Edit Ellipse FileBox Font Form Grid Group HTMLRenderer Icon Image ImageList Label List ListView Locator Marker MDIClient Menu MenuBar Metafile MsgBox OCXClass OLEClient OLEServer Poly Printer ProgressBar PropertySheet Rect RichEdit Scroll Spinner Splitter Static StatusBar SubForm TabBar TabControl TCPSocket Text Timer TipField ToolBar ToolControl TrackBar TreeView UpDown

Properties Type ClassName Posn Size Coord Border Active Visible Event Draggable FontObj FCol BCol Picture CursorObj AutoConf YRange XRange Data TextSize EdgeStyle Handle Translate Accelerator AcceptFiles ClassID Container KeepOnClose HelpFile ToolboxBitmap TypeLibID TypeLibFile LastError Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj SetFnInfo SetVarInfo SetEventInfo

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap AmbientChanged Select PreCreate

Description

The ActiveXControl object represents a Dyalog APL namespace as an ActiveX control.

During development, an ActiveXControl is a container object that is the child of a Form and acts as a wrapper for one or more other GUI objects.

To make an ActiveXControl available to another application, you must select *Make OCX* from the Session *File* menu. This creates an .OCX file that contains your entire workspace and all of the ActiveXControls within it.

Once an ActiveXControl has been saved in an .OCX file, any application that supports ActiveX may create and use instances of it.

When an ActiveX control is loaded by a host application, it and any code that it requires, is loaded into the host application's address space; it does not run in a separate address space.

During development, an ActiveXControl is powered by the development version of Dyalog APL. However, an ActiveXControl object that is loaded by a host application, is powered by a DLL version of Dyalog APL. This automatically gets loaded when a host application creates the first instance of any Dyalog APL ActiveX control. However, within a single host application, other instances of the same or other Dyalog APL ActiveX controls share the same copy of DYALOG.DLL.

Like the development and run-time versions of Dyalog APL, DYALOG.DLL has an active workspace. When an application loads an ActiveXControl, DYALOG.DLL *copies* the *top-level namespace* that owns the ActiveXControl, together with everything it contains, into the active workspace. For example, if the ActiveXControl is named `Controls.Form1.Ctrl1`, the act of creating the first instance of `Ctrl1` will cause the entire contents of the `Controls` namespace to be copied, from the corresponding .OCX file, into the active workspace. This affords the potential for controls from different OCX files to clash, but the name clash conflict is restricted to just one name.

Each instance of an ActiveXControl, is represented by a separate namespace which is automatically *cloned* from the original ActiveXControl namespace. Each instance namespace is entirely separate from any other instance namespace and there is no way for one instance to reference or see any other instance; nor can it reference the original class namespace from which it was cloned. In fact, each instance appears to itself to be the one and only original class namespace. Using the previous example, each instance of `Ctrl1` believes that its full pathname is `#.Controls.Form1.Ctrl1`, although each instance is in fact a separate clone of that namespace.

When an application creates an instance of an ActiveXControl, it does so as the child of some object within its own GUI hierarchy. From the instance's viewpoint, its parent Form is replaced by a different GUI object that imposes position, size, font, background colour, and other ambient properties.

The external name of an ActiveXControl is made up of the character vector defined by the `ClassName` property, prefixed by the string "Dyalog ", and followed by the string "Control". If `ClassName` is empty (which is the default), the name of the ActiveXControl namespace is inserted instead. Note that the name should not include APL symbols such as `Δ.ClassName` may only be specified when you create the ActiveXControl with `⌶WC` and may not be changed using `⌶WS`.

The *Coord* property is read-only and its value is always 'Pixel'. If you wish to use a different co-ordinate system for the children of an ActiveXControl object, it is necessary to set *Coord* separately on each one of them.

Posn and *Size* are negotiable properties. When an instance of the ActiveXControl is created, the values of *Posn* and *Size* will be assigned by the host application. You may change these values using `OWS`, but the host application has the right to refuse them and there is no guarantee that you will get what you set.

The *Border* and *EdgeStyle* properties may be used to control the outline appearance of the ActiveXControl object.

The *Dragable* and *KeepOnClose* properties apply only during development and are otherwise ignored.

The *ToolboxBitmap* property specifies the name of a *Bitmap* object that may be used by a host application to represent the ActiveXControl when its complete visual appearance is not required.. For example, if you add an ActiveX control to the Microsoft Visual Basic development environment, its bitmap is added to the toolbox. The Bitmap should therefore be of an appropriate size, usually 24 x 24 pixels.

The *Container* property provides access to an *ActiveXContainer* object that represents the host application itself. This may be used to obtain the values of ambient properties, or to access methods exposed by the host application via OLE interfaces.

When an instance of an ActiveXControl is created, it generates first a *PreCreate* event, and then a *Create* event. The *PreCreate* event is generated at the point the *instance* is made.

The *Create* event is generated at the point when the host application requires the instance to appear visually. If, as is recommended, you create child controls of the instance when it is created, you must respond to the *Create* event, because at the time that *PreCreate* is generated, the object does not have a window.

Host applications which support two different modes of operation, namely design mode and run mode, differ in the way that they create instances of ActiveX controls. Microsoft Access does not require an ActiveX control to appear properly in design mode. Instead, it draws a simple box containing just the name of the object. If your ActiveXControl is hosted by Microsoft Access, it will get a *PreCreate* Event when an instance is created in design mode, and a *Create* event only when it enters run mode. Microsoft Visual Basic, however, requires the object to draw itself immediately, even in design mode, and so a *Create* event will be generated immediately after a *PreCreate* event in this case.

2.3 Animation Object

Purpose: The Animation object displays simple animations from basic .AVI files or resources.

Parents ActiveXControl Form Group PropertyPage SubForm

Children Bitmap Circle Cursor Ellipse Font Marker Poly Rect Text Timer

Properties Type Posn Size File Coord Border Active Visible Event Sizeable Dragable BCol AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Translate AcceptFiles KeepOnClose AutoPlay Transparent Align MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj AnimOpen AnimClose AnimPlay AnimStop

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap AnimStarted AnimStopped

Description

The Animation object displays simple animations from basic .AVI files or resources.

The Animation object can only play AVI files or resources that have no sound and can only display uncompressed AVI files or .AVI files that have been compressed using Run-Length Encoding (RLE).

For more sophisticated animations, you may use the Windows Media Player (OCX).

To display an AVI file, you must first use the *AnimOpen* method to open it. If the *AutoPlay* property is set to 1, the animation will play immediately. Otherwise, only the first frame will be displayed.

The *Align* property may be 'None' or 'Centre' ('Center'). If *Align* is 'None', the Animation window is automatically resized to fit the AVI being played. If *Align* is 'Centre', the AVI is centred in the Animation window. If the window is too small, the AVI is clipped.

The *AnimPlay* method may be used to play the animation and allows you to specify the start, number of frames, and repeat count.

The *AnimStop* method causes the animation to stop.

The *AnimClose* method closes the current AVI file and resets the contents of the object's window to its background colour.

The *AnimStarted* and *AnimStopped* events are reported when the animation starts and stops respectively.

2.4 Bitmap Object

Purpose: A graphical object used to represent a bitmap which may be used both to display a picture or as a pattern (brush) used to fill other objects.

Parents ActiveXControl Animation Button CoolBand Form Grid Group ImageList ListView Menu MenuBar MenuItem NetType OLEServer Printer ProgressBar PropertyPage PropertySheet RichEdit Root StatusBar SubForm TCPSocket ToolBar ToolButton ToolControl TrackBar TreeView UpDown

Children Circle Ellipse Font Image Marker Metafile Poly Rect Text Timer

Properties Type File Bits CMap KeepBits Size Coord Event FontObj YRange XRange Data TextSize Translate Accelerator KeepOnClose CBits MaskCol MethodList ChildList EventList PropList

Methods Detach FileRead FileWrite MakePNG MakeGIF GetTextSize

Events Close Create Select

Description

A Bitmap may be created either from a file (.BMP, .GIF or .PNG) or from APL arrays. To create a Bitmap object using *Dyalog APL Language: Wc*, you can either specify the *File* property **or** the *CBits* property, **or** the *Bits* and *CMap* properties.

If you specify *File*, it should contain the name of a bitmap file from which the bitmap is to be read. If omitted, a .BMP file extension is added. You may also load a Bitmap from a DLL or from the DYALOG.EXE executable. See *File* property for details.

If instead you want to create a Bitmap dynamically from APL variables, you may do so in one of two ways.

For a palette of up to 256 colours, you may specify the image using the *Bits* and *CMap* properties. The alternative is to use the *CBits* property which works for any size of colour palette.

If *MaskCol* is non-zero, it specifies the transparent colour for the Bitmap. Any pixels specified with the same colour will instead be displayed in whatever colour is underneath the Bitmap. This achieves similar behaviour to that of an Icon.

The *KeepBits* property has the value 0 or 1, and controls how a Bitmap is saved in the workspace. A value of 0 (the default) means that the values of *CBits*, *Bits* and *CMap* are **not** kept in the workspace. If you request the values of *CBits*, *Bits* or *CMap* with *Dyalog APL Language: Wg*, they are obtained directly from the corresponding Windows bitmap resource. When the workspace is `)LOADED`, the Bitmap is recreated from the associated file defined by the value of the *File* property. Note that if this file doesn't exist when the workspace is `)LOADED`, the Bitmap is not created, but no error is generated. However, when you reference the object you will get a `VALUE ERROR`.

If *KeepBits* is 1, the values of *CBits*, *Bits* and *CMap* **are** stored permanently in the workspace, and are used to rebuild the Bitmap when the workspace is `)LOADED`. In this case, the file name (if any) is ignored. Setting *KeepBits* to 1 uses more workspace, but may be more convenient if you want to distribute applications.

The *Size* property allows you to query the size of a Bitmap without having to retrieve the *CBits* or *Bits* property and then take its "shape". This will be noticeably faster for a large Bitmap. If you set the *Size* property using *Dyalog APL Language: Ws* the Bitmap is scaled to the new size.

A useful feature of a Bitmap is that it can be the parent of any of the graphical objects. This allows you to create or edit a bitmap by drawing lines, circles, etc. in it.

The *FileRead* (90) and *FileWrite* (91) methods allow you to dynamically manage bitmap files (.BMP). The expression:

```
bmname.FileWrite
```

causes the Bitmap called *bmname* to be written to the file specified by the current value of the *File* property. The file is automatically written in standard bitmap format. Similarly, the expression:

```
bmname.FileRead
```

causes the Bitmap called *bmname* to be redefined from the bitmap file specified by the current value of the *File* property.

The *MakeGIF* and *MakePNG* methods may be used to convert the image represented by a Bitmap object into a GIF or PNG data stream, suitable for display in a web browser. The *TCPSendPicture* method may be used to transfer a Bitmap on a TCP/IP socket.

Using a bitmap is always a 2-stage process. First you create a Bitmap object with *Dyalog APL Language: Wc*. Then you use it by specifying its name as a property of another object.

The *Picture* property specifies the name of a Bitmap to be displayed in an *ActiveXControl*, *Button*, *Form*, *Group*, *Image*, *MDIClient*, *Sm*, *Static*, *StatusBar*, *StatusField*, *SubForm*, *TabBar*, or *ToolBar*.

The *BtnPix* property specifies three Bitmaps to be used to represent the 3 states of a *Button*, *Menu* or *MenuItem*.

The *FStyle* property specifies the name of a Bitmap to be used as a pattern to fill a *Poly*, *Ellipse* or *Rect* object.

2.5 BrowseBox Object

Purpose: The BrowseBox object allows the user to browse for and select a folder or other resource.

Parents ActiveXControl CoolBand Form Grid OLEServer PropertyPage PropertySheet Root StatusBar SubForm TCPSocket ToolBar ToolControl

Children Timer

Properties Type Caption BrowseFor Target StartIn HasEdit Event Data Translate KeepOnClose MethodList ChildList EventList PropList

Methods Detach Wait

Events Close Create FileBoxOK FileBoxCancel

Description

The BrowseBox object is a dialog box that allows the user to browse for and select a folder (directory) or other resource.

For full functionality as described here, the BrowseBox object requires the Windows Shell Library SHELL32.DLL Version 4.71 or higher. The BrowseBox object also supports the enhanced functionality provided by SHELL32.DLL Version 5 (Windows 2000) if present.

The *BrowseFor* property specifies the type of resource and may be 'Directory' (the default), 'File', 'Computer' or 'Printer'.

The *StartIn* property specifies the path name where browsing should start.

The *HasEdit* property specifies whether or not the dialog box contains an edit field into which the user can type the name of the folder or other resource, rather than browsing for it. The default is 0.

A BrowseBox may only be used by the execution of a modal □□□. The action code for the FileBoxOK and FileBoxCancel events must be set to 1 so that the appropriate result is returned by the modal □□□.

After the user has pressed OK or Cancel, the *Target* property contains the name of the chosen folder or other resource.

EXAMPLE

```

▼ DIR←{START_DIR}GetDir CAPTION;BB;MSG
[1]   A Ask user for a Directory name
[2]   A CAPTION specifies Caption for dialog box
[3]   A START_IN (optional) specifies starting directory
[4]   A DIR is empty if user cancels
[5]   :With 'BB'□□WC'BrowseBox'
[6]       :If 2=□□NC'START_DIR'
[7]           StartIn←START_DIR
[8]       :Else
[9]           StartIn←''
[10]      :EndIf
[11]      onFileBoxOK←onFileBoxCancel←1
[12]      Caption←CAPTION
[13]      HasEdit←1
[14]      MSG←□□□□''
[15]      :If 'FileBoxOK'≡2>MSG
[16]          DIR←Target A = 3>MSG
[17]      :Else
[18]          DIR←''
[19]      :EndIf
[20]   :EndWith
▼

```

2.6 Button Object

Purpose: Allows the user to initiate an action or to select an option using a button.

Parents ActiveXControl CoolBand Form Grid Group PropertyPage SubForm ToolBar
ToolControl

Children Bitmap Circle Cursor Ellipse Font Marker Poly Rect Text Timer

Properties Type Caption Posn Size Style Coord Align State Default Cancel Border Justify Active Visible Event Sizeable Dragable FontObj FCol BCol Picture BtnPix CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj ReadOnly Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex Elevated Note MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ChooseFont

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap FontOK FontCancel Select DropDown

Description

The type of button displayed is determined by the *Style* property which may take the value 'Push', 'Radio', 'Check', 'Toggle', 'Split' or 'CommandLink'. Under Windows, 'Toggle' and 'Check' are treated identically.

'Split' and 'CommandLink' apply only to Windows Vista and later and require *Native Look and Feel*

. Otherwise the use of these Styles will produce a Button with Style 'Push'.

Push buttons are used to generate actions. When the user "presses" a pushbutton, it generates a *Select* event (30). To cause an action, you simply associate the name of a callback function with this event for the Button in question.

Radio buttons and Check boxes are used to select options. They each have two states between which the user can toggle by clicking the mouse. When the Button (option) is selected, its *State* property has the value 1; otherwise it is 0.

Only one of a group of Radio buttons which share the same parent can be set (*State* is 1) at any one time. Radio buttons are therefore used for a set of choices that are **mutually exclusive**. Check boxes however, may be set together to signify a combination of options. These are used for making choices which are not mutually exclusive.

Radio and Check buttons **also** generate *Select* events when their *State* changes, and you can attach callback functions to these events to keep track of their settings. However, as Radio and Check buttons are not normally used to generate actions, it is perhaps easier to wait until the user signifies completion of the dialog box in some way, and then query the *State* of the buttons using *Dyalog APL Language: Wg*. For example,

if you have a set of Radio or Check buttons in a *Group* called `f1.options`, the following statements retrieve their settings.

```
OPTIONS ← (□WN 'f1.options') □WG''c 'State'
```

or

```
OPTIONS ← □WG'State' '' □WN 'f1.options'
```

The *Caption* property determines the text displayed in the Button. Its default value is an empty vector. If the *Caption* property contains one or more linefeed characters (□UCS 10) the text is centre-justified (*Style* 'Push' only) or top-left justified and automatically wraps on white-space characters (such as space and tab) to fit in the width provided.

If *Style* is 'Radio' or 'Check', the text may be aligned to the left or right of the button graphic using the *Align* property. Its default value is 'Right'.

The CommandLink button has an icon displayed to the left of its *Caption*. The appearance of the icon is controlled by the *Elevated* property. *Elevation* is a feature of *User Account Control* in Windows 7. In addition to the *Caption*, additional text may be defined by its *Note* property. If provided, this is displayed below the *Caption*.

The Split *Button* has a drop-down button, similar to that provided by a *Combo* object.

If *Posn* is omitted, the button is placed in the centre of its parent. If either element of *Posn* is set to 0, the button is centred along the corresponding axis.

If *Size* is not specified, the size of the button is determined by its *Caption*. If either element of *Size* is set to 0 the corresponding dimension is determined by the height or width of its *Caption*. This does not apply to multi-line buttons whose dimensions should be specified explicitly. If *Caption* is not specified, or is set to an empty vector, the value of *Size* is set to a default value.

Button colours can be specified using *FCol* and *BCol*. However, pushbuttons (*Style* 'Push') **ignore** *BCol* and instead use the standard Windows colour.

The *Picture* property is used to display a bitmap on a pushbutton. This property is a 2-element array containing the name of a *Bitmap* object and the "mode" in which it is to be displayed. The default mode (3) is the most useful, as it causes the *Bitmap* to be **superimposed** on the centre of the Button. The surrounding edges of the Button (which gives it its 3-dimensional appearance and pushbutton behaviour) are unaffected. Note that if *Picture* is set on a Button whose *Style* is 'Radio' or 'Check', the Button assumes pushbutton appearance, although its 'Radio' / 'Check' behaviour is preserved.

An alternative is to use the *BtnPix* property. This property specifies the names of up to 3 *Bitmap* objects. The first *Bitmap* is displayed when the *State* of the Button is 0. The second is displayed when its *State* is 1. The third is shown when the Button is inactive (*Active* 0). *BtnPix* is more flexible than *Bitmap*, but if you want your Button to exhibit pushbutton behaviour, you must design your bitmap accordingly.

The *ReadOnly* property is Boolean and specifies whether or not the user may change the state of the Button. It applies only to *Style* 'Radio' and *Style* 'Check'.

The user can interact with the *Button* by clicking it, which generates a *Select* Event or (*Style* 'Split') the drop-down which generates a *DropDown* Event.

2.7 ButtonEdit Object

Purpose: Allows user to enter or edit data.

Parents ActiveXControl Form Group PropertyPage SubForm

Children Circle Ellipse Font ImageList Marker Poly Rect Text Timer

Properties Type Text Posn Size Style Coord Align Border Justify Active Visible Event ImageListObj SelText Sizeable Dragable FontObj FCol BCol CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj FieldType MaxLength Decimals Password ValidIfEmpty ReadOnly FormatString Changed Value Translate Accelerator AcceptFiles KeepOnClose Transparent ImageIndex Redraw TabIndex Cue ShowCueWhenFocused HasClearButton MethodList ChildList EventList PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap Select BadValue KeyError Change DropDown

Description

The ButtonEdit object combines a single-line input field with a customisable button. It provides the same user and programmer interfaces as an *Edit* object (*Style* 'Single').

The appearance of the button, which is displayed to the right of the input field, is determined by the *ImageListObj* property. When clicked, the object generates a

DropDown event. There is no default processing for this event; it is up to the programmer to take the appropriate action via a callback function.

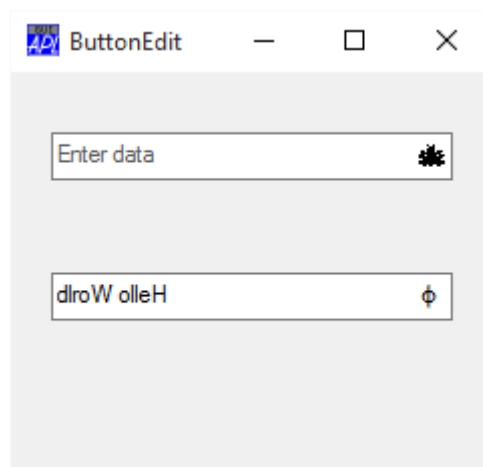
The following picture illustrates two ButtonEdit objects

```

▼ Example;BK;White;dialog
[1]  dialog←2 □NQ'.' 'GetEnvironment' 'Dialog'
[2]  'F'□WC'Form' 'ButtonEdit'('Coord' 'Pixel')('Size' 200 240)
[3]  'F'□WS'Coord' 'Pixel'
[4]  'F.IL1'□WC'ImageList'('Size' 16 16)('Masked' 1)
[5]  'F.IL1.Time'□WC'Icon'(dialog,'\\ws\\arachnid.ico')
[6]  'F.BE1'□WC'ButtonEdit' ''(30 20)(θ 200)
[7]  F.BE1.(Cue ShowCueWhenFocused)←'Enter data' 1
[8]  F.BE1.(ImageListObj ImageIndex)←F.IL1 1
[9]
[10] 'F.fnt'□WC'Font' 'APL385 Unicode' 16
[11] BK←16 16p256⊥White←255 255 255
[12] 'F.Rotate'□WC'Bitmap'('CBits'BK)('MaskCol'White)
[13] 'F.Rotate.'□WC'Text' 'φ'(0 3)('FontObj'F.fnt)
[14] BK←F.Rotate.CBits
[15] 'F.IL1.'□WC'Bitmap'('CBits'BK)('MaskCol'White)
[16] 'F.BE2'□WC'ButtonEdit' 'Hello World'(100 20)(θ 200)
[17] F.BE2.(ImageListObj ImageIndex)←F.IL1 2
[18] F.BE2.onDropDown←'Rotate'
▼

▼ Rotate msg
[1]  (≻msg).Text←φ(≻msg).Text
▼

```



Note

For full functionality (in particular, for the *Cue* property to apply), the ButtonEdit object requires that *Native Look and Feel* is enabled.

2.8 Calendar Object

Purpose: The Calendar object provides an interface to the Month Calendar Control

Parents ActiveXControl Form Group PropertyPage SubForm ToolBar

Children Cursor Font Menu MsgBox TCP Socket Timer

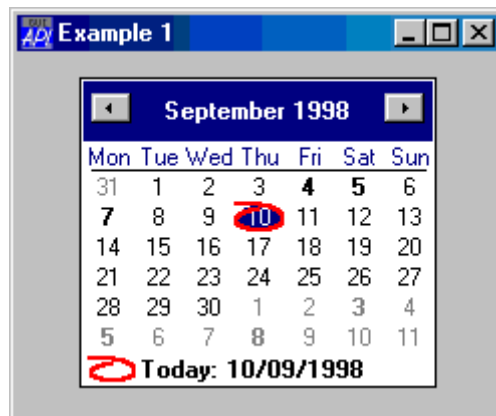
Properties Type Posn Size Style Coord Border Active Visible Event FirstDay MaxSelCount SelDate MinDate MaxDate CalendarCols Today HasToday CircleToday WeekNumbers MonthDelta Sizeable Draggable FontObj CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Accelerator KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj GetVisibleRange GetMinSize IDNToDate DateToIDN

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDblClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap SelDateChange GetDayStates CalendarDown CalendarUp CalendarDblClick CalendarMove Select

Description

The Calendar object displays a calendar and allows the user to select a date or range of dates. The following illustration shows a default Calendar object.



The Calendar object will display as many full months as it can fit into the area specified by its Size property as shown below. The minimum size required to encompass a single month may be obtained using the *GetMinSize* method.

```
'F'WC'Form' 'Example 2'('Size' 50 50)
'F.C'WC'Calendar'('Size' 100 100)
```



The *Today* property is an *IDN* that specifies the current day. Its default value is today's date, that is, the local date set on your computer.

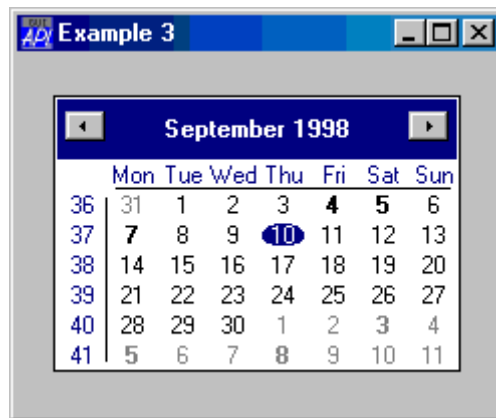
The *CircleToday* property is either 0 or 1 (the default) and specifies whether or not the Today date is circled when the Calendar object is showing the corresponding month.

The *HasToday* property is either 0 or 1 (the default) and specifies whether or not the Today date is displayed (using the Windows short date format) in the bottom left of the Calendar object.

The *WeekNumbers* property is either 0 (the default) or 1 and specifies whether or not the Calendar displays week numbers.

The following example shows a Calendar with both *CircleToday* and *HasToday* set to 0 and *WeekNumbers* set to 1.

```
'F'WC'Form' 'Example 3'('Size' 30 30)
'F.C'WC'Calendar'('CircleToday' 0)('HasToday' 0)('WeekNumbers' 1)
```



The *FirstDay* property is an integer whose value is in the range 0-6. *FirstDay* specifies the day that is considered to be the first day of the week and which appears first in the Calendar. The default value for *FirstDay* depends upon your International Settings.

The *MinDate* and *MaxDate* properties are integers that specify the minimum and maximum *IDN* values that the user may display and select in the Calendar object. By default these properties specify the entire range of dates that the Windows Month Calendar control can provide.

The *MonthDelta* property specifies the number of months by which the Calendar object scrolls when the user clicks its scroll buttons. The default is empty (zilde) which implies the number of months currently shown.

The *Style* property may be either 'Single' (the default) or 'Multi'. If *Style* is 'Single', the user may select a single date. If *Style* is 'Multi', the user may select a

contiguous range of dates. In this case, the maximum number of contiguous days that can be selected is defined by the *MaxSelCount* property which is an integer whose default value is 7.

The *SelDate* property is a 2-element integer vector of *IDN* values that identifies the first and last dates that are currently selected.

When the user selects one or more dates, the Calendar object generates a *SelDateChange* event. This event is also generated when the Calendar object is scrolled, and the selection changes automatically to another month.

The Calendar displays day numbers using either the normal or the bold font attribute and you may specify which attribute is to be used for each day shown. However, the Calendar object does not store this information beyond the month or months currently displayed.

When the Calendar control scrolls (and potentially at other times), it generates a *GetDayStates* event that, in effect, asks you (the APL program) to tell it which (if any) of the dates that are about to be shown should be displayed in bold.

If you wish *any* dates to be displayed using the bold font attribute, you must attach a callback function to the *GetDayStates* event which returns this information in its result. By default, all dates are displayed using the normal font attribute, so you only need a callback function if you want any dates to be displayed in bold.

You may also set the font attribute for particular days in the range currently displayed by calling *GetDayStates* as a method.

The *CalendarCols* property specifies the colours used for various elements in the Calendar object.

You may convert dates between *IDN* and *ITS* representations using the *IDNToDate* and *DateToIDN* methods. Note that these methods apply to **all** objects and not just to the Calendar object itself.

The *GetVisibleRange* method reports the range of dates that is currently visible in the Calendar object.

2.9 Circle Object

Purpose: A Graphical object to draw circles, arcs, and pie-slices.

Parents ActiveXControl Animation Bitmap Button ButtonEdit Combo ComboEx Edit Form Grid Group Label List ListView MDIClient Metafile Printer ProgressBar PropertyPage PropertySheet RichEdit Scroll Spinner Static StatusBar SubForm TabBar TipField ToolBar TrackBar TreeView UpDown

Children Timer

Properties Type Points Radius FCol BCol Start End ArcMode LStyle LWidth FStyle FillCol Coord Visible Event Dragable OnTop CursorObj AutoConf Data Accelerator KeepOnClose DrawMode RadiusMode MethodList ChildList EventList PropList

Methods Detach

Events Close Create DragDrop MouseDown MouseUp MouseMove MouseDbClick Help Select

Description

The *Points* property contains the co-ordinates of the centre of the circle. The size of the circle is determined by the *Radius* property. This specifies the radius along the x-axis, the height is calculated so that the object is circular.

The *RadiusMode* property determines whether or not the circle is adjusted by a pixel, if required in order to appear perfectly round and perfectly centred. The default value is 0 (no adjustment is made).

The *Start* and/or *End* properties are used to draw partial circles. They specify start and end angles respectively, measuring from the x-axis in a counter-clockwise direction and are expressed in radians. The type of arc is controlled by *ArcMode* as follows :

ArcMode Effect

0	An arc is drawn from <i>Start</i> to <i>End</i> .
1	An arc is drawn from <i>Start</i> to <i>End</i> . In addition, a single straight line is drawn from one end of the arc to the other, resulting in a segment.
2	An arc is drawn from <i>Start</i> to <i>End</i> . In addition, two lines are drawn from each end of the arc to the centre, resulting in a pie-slice

.

Points, *Radius*, *Start* and *End* can specify vectors so that several arcs, circles, pie slices, etc. can be drawn in one call (and with one name).

If *Start* is specified, but not *End*, end angles default to $(-1 \div \text{Start}) \div 2$. If *End* is specified, but not *Start*, start angles default to 0, $-1 \div \text{End}$

This means that you can draw a pie-chart using either *Start* or *End* angles; you do not have to specify both.

EXAMPLES

A circle whose centre is (50,50) and radius 20

```
'g.p1' WC 'Circle' (50 50) 20
```

An arc

```
'g.arc' WC 'Circle' (50 50) 20 ('Start' (0.75)) ('End' (1.25))
```

Complete pie

```
Data←12 27 21 40
ANGLES←0, -1÷((0.2)÷÷/Data)×÷\Data
COLS←(255 0 0)(0 255 0)(255 255 0)(0 0 255)
PATS←1 2 3 4
'g.pie' WC 'Circle' (50 50) 20 ('Start' ANGLES)
      ('ArcMode' 2) ('FCol' (0 0 0))
      ('FStyle' PATS) ('FillCol' COLS)
```

Same pie as above, but 2nd slice is exploded by changing its centre and 4th slice is shrunk by reducing its radius :

```
CY←50 52 50 50  # y-coord of centres
R←20 20 20 17.5 # radii
'g.pie' WC 'Circle' (50 CY) R ('Start' ANGLES)
      ('ArcMode' 1) ('FCol' (0 0 0))
      ('FStyle' PATS) ('FillCol' COLS)
```

2.10 Clipboard Object

Purpose: This object provides access to the Windows clipboard.

Parents ActiveXControl CoolBand Form OLEServer PropertyPage PropertySheet Root TCPSocket

Children Timer

Properties Type Event Data Formats Text Bits CMap CBits MetafileObj Picture Array
RTFText Translate Accelerator KeepOnClose MethodList ChildList EventList PropList

Methods Detach Wait

Events Close Create ClipChange Select

Description

When an application places data in the Windows clipboard, it may store it in one or more formats. An application wishing to retrieve data from the clipboard can then choose which format to read it in. Dyalog APL supports standard clipboard formats, including CF_TEXT, CF_BITMAP and CF_METAFILE. If there is any data in the clipboard, the *Formats* property lists the formats in which it may be retrieved.

In addition, the *Array* property may be used to set or retrieve clipboard contents in Dyalog APL array format.

Data is read from the clipboard using *Dyalog APL Language: Wg*, specifying the name of the appropriate property for the data that you want.

If the data has been stored in CF_Text format, the value of *Formats* will include 'Text' and you may retrieve the data by querying the value of the *Text* property with *Dyalog APL Language: Wg*.

If the data has been stored in **device-independent** bitmap format, the value of *Formats* will include 'CBits', 'Bits' and 'CMap'. To retrieve the bitmap pattern and colour map, you may query the values of the *CBits*, or *Bits* and *CMap* properties using *Dyalog APL Language: Wg*.

If the data has been stored in **device-dependent** bitmap format, only the bitmap pattern is available and *Formats* will contain 'Bits' but not 'CMap'. In this case you can query the *Bits* property but not *CMap* without which you cannot realise the bitmap. However, if data was posted in this format, it is highly probable that the current Windows colour map applies to it. For a standard 16-colour device this is given under the description of the *CMap* property.

The following example retrieves text from the clipboard :

```
'CL' □WC 'Clipboard'
Data ← 'CL' □WG 'Text'
```

The next example retrieves a bitmap from the clipboard and defines it as a *Bitmap* object named 'BM' ready for use :

```
'BM' □WC 'Bitmap' '', 'CL' □WG 'Bits' 'CMap'
```

Data may be placed in the clipboard using *Dyalog APL Language: Wc* or *Dyalog APL Language: Ws*. To store text, you simply set the *Text* property. You may use a simple character vector or matrix, or a vector of character vectors. For example :

```
'CL' □WS 'Text' 'Hello World'
```

To store a bitmap you can set either the *Picture* property to the **name** of a *Bitmap* object, or you can set the *Bits* and *CMap* properties explicitly. The former is more efficient, especially for large bitmaps, for example :

```
'CL' □WS 'Bitmap' 'BM'
```

or

```
Bits CMap ← 'BM' □WG 'Bits' 'CMap'
'CL' □WS ('Bits' Bits)('CMap' CMap)
```

Note that if you use the latter method, you must set **both** properties in one *Dyalog APL Language: Ws* statement. This is also true if you wish to store data in both Text and Bitmap formats together.

The *Metafile* property allows graphical information to be restored in and retrieved from the clipboard in Windows Metafile format. See the description of the *Metafile* property for details.

A *ClipChange* (120) event is generated when another application places data in the clipboard.

2.11 ColorButton Object

Purpose: The ColorButton object allows the user to select a colour.

Parents ActiveXControl Form Grid Group PropertyPage SubForm

Properties Type Caption Posn Size CurrentColor DefaultColors CustomColors OtherButton Coord Active Visible Event Sizeable Dragable CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator KeepOnClose ShowDropDown Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap ColorChange DropDown

Description

The *ColorButton* object displays a coloured box, with an optional drop down button. When the user clicks the *ColorButton* with the left mouse button, a colour selection drop-down appears below it, allowing the user to select a new colour.

The *CurrentColor* property (default 0 0 0) is a 3-element integer vector that specifies and reports the RGB value of the currently selected colour.

The *DefaultColors* property is a nested matrix which specifies the RGB values of the colours shown in the colour selection box. The shape of *DefaultColors* determines the number of rows and columns in the colour selection drop-down. Each element of *DefaultColors* is a 3-element integer vector specifying an RGB colour value.

The *OtherButton* property is Boolean and specifies whether or not the user can select a colour using the Windows colour selection dialog box.

If *OtherButton* is 1 (the default), the final row of the colour selection drop-down contains a button labelled "Other...". If the user clicks this button, the standard Windows colour selection dialog box is displayed, allowing the user to select any colour that the computer can render.

If *OtherButton* is 0, the button labelled "Other..." is not present and the user is restricted to the choice of colours provided by the *DefaultColors* property.

The *CustomColors* property is a 1-row, 16-column nested matrix which specifies the RGB values of the Colours displayed in the *Custom colors* section of the Windows colour selection dialog box. Each element of *CustomColors* is a 3-element integer vector specifying an RGB colour value.

The *ShowDropDown* property is Boolean (default 1) and specifies whether or not a drop-down button is displayed in the *ColorButton* object.

When the user clicks a *ColorButton* with the left mouse button, the object generates a *DropDown* event just before it displays the colour selection drop-down. This event may be used to set the *DefaultColors* and/or *CustomColors* properties dynamically.

When the user selects a new colour, the *ColorButton* generates a *ColorChange* event.

2.12 Combo Object

Purpose: This object combines an input area with a list box and allows the user to enter a selection by typing text or by choosing an item from the list.

Parents ActiveXControl CoolBand Form Grid Group PropertyPage SubForm ToolBar ToolControl

Children Circle Cursor Ellipse Font Marker Poly Rect Text Timer

Properties Type Items Text Posn Size Style Coord Rows Border Active Visible Event VScroll HScroll SelItems SelText Sizeable Dragable FontObj FCol BCol CursorObj AutoConf Index Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex HasClearButton MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ChooseFont

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap DropDown Select Change

Description

Three types of Combo box are provided by the *Style* property which may be 'Drop' (the default), 'Simple' or 'DropEdit'.

The *Items* property specifies the list of items which are displayed in the list box and from which the user can choose.

The *SelItems* property is a Boolean vector which specifies which (if any) of the items is selected. When the user chooses an item from the list, it is copied to the edit field and a *Select* event is generated. At this point you may use *SelItems* to identify the chosen item. You can also use *SelItems* to pre-select the contents of the edit field.

If the *Style* is 'Simple' or 'DropEdit', the user may type text into the edit field. In these cases, the contents of the edit field may also be specified or queried using the *Text* property.

Note that if the user first selects an item from the list box, then changes it in the edit field, the entry in the list box is automatically deselected. There is therefore no conflict between the value of *Text* and the value of *SelItems*.

Warning

Windows truncates the contents of the edit field (reflected in the value of the `Text` property) to 510 characters.

For a Combo with `Style 'Simple'`, the `Index` property specifies or reports the position of `Items` in the list box as a positive integer value. If `Index` has the value "n", it means that the "nth" item in `Items` is displayed on the top line in the list box. Note that `Index` can only be set using *Dyalog APL Language: Ws* and **not** by *Dyalog APL Language: Wc* and is ignored if all the `Items` fit in the list box. The default value for `Index` is `⌊IO`.

The `SelText` property identifies the portion of the edit field that is highlighted. It is not applicable to a Combo with `Style 'Drop'` as the user cannot enter or change data in its edit field.

The height of a Combo object with `Style 'Drop'` or `'DropEdit'` is defined in a manner that is different from other objects. The height of the edit field is fixed, and is dependent only upon the size of the font. The height of the associated drop-down list box is determined by the `Rows` property. The first element of the `Size` property (height) is ignored. For a Simple combo box (whose listbox is permanently displayed), the overall height is determined by the first element of `Size`. `Rows` is a "read-only" property.

The `VScroll` property specifies whether or not a vertical scrollbar is provided. The default value 0 means no scrollbar, setting `VScroll` to `-1` or `-2` specifies that the Combo has a vertical scrollbar.

If the `Style` is `'Simple'` or `'DropEdit'`, the `HScroll` property determines whether or not the edit field may be scrolled. If `HScroll` is 0, the data is not scrollable, and the user cannot enter more characters once the field is full. If `HScroll` is `-1` or `-2` the field is scrollable, and there is no limit on the number of characters that can be entered. In neither case however is a horizontal scrollbar provided. If `Style` is `'Drop'`, the user is not allowed to enter data into the edit field anyway, and the value of `HScroll` is ignored.

`VScroll` and `HScroll` may only be set when the object is created and may not subsequently be changed.

Note that when you change the `Items` property using *Dyalog APL Language: Ws*, the `Text`, `SelItems` and `SelText` properties are all reset to their default values.

The Combo object will report a *Select* event (if enabled) when the user chooses an item from the list box. It will generate a *Change* event (if enabled) when the user manually alters the contents of the edit field and then changes the focus to another object.

2.13 ComboEx Object

Purpose: The ComboEx object is an extended version of the Combo object that provides additional features including item images

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm ToolBar ToolControl

Children Circle Cursor Ellipse Font Marker Poly Rect Text Timer

Properties Type Items Text Posn Size Style Coord Rows Border Active Visible Event Indents ImageListObj ImageIndex SellImageIndex CaseSensitive EditImage EditImageIndent PathWordBreak VScroll HScroll SellItems SelText Sizeable Draggable FontObj FCol BCol CursorObj AutoConf Index Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex HasClearButton MethodList ChildList EventList PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap DropDown Select

Description

The ComboEx object is a ComboBox that supports item images and indenting. It is a superset of the *Combo* object and supports all its functionality. For further details, see "*Combo*".

For most purposes, you can use the ComboEx object in place of the *Combo* object whether or not you make use of the extended features of the ComboEx.

Like the basic *Combo*, the list of text items in the ComboEx is specified by the *Items* property. You may associate images with each of the text items using the *ImageListObj*, *ImageIndex* and *SellImageIndex* properties.

To do so, *ImageListObj* specifies the name of an *ImageList* object that contains a set of images. *ImageIndex* and *SellImageIndex* map individual images from the *ImageList* to each of the text items specified by *Items*. *ImageIndex* specifies the image to be displayed when the item is not selected; *SellImageIndex* specifies the image to be displayed when the item is selected.

The *Indents* property specifies the amount by which each of the items are indented in units of 10 pixels

The appearance of the items is additionally controlled by the *EditImage* and *EditImageIndent* properties. These are Boolean and their effect is summarised in the table below. Notice that Images are displayed only if both these properties are set to 1 (which is the default).

There are certain restrictions that apply to a *ComboBox* object with Style 'Simple', namely:

- images and indents do not apply to the edit control portion of the object.
- the object may not redraw properly if *EditImage* and/or *EditImageIndent* are set to 0 or if *CaseSensitive* or *PathWordBreak* are set to 1.
- *PathWordBreak* does not work.

	EditImageIndent	
EditImage	0	1
0	No images displayed, item text is indented as specified by <i>Indents</i>	No images displayed, item text is indented as specified by <i>Indents</i> plus the width of the images in <i>ImageList</i>
1	No images displayed, item text is indented as specified by <i>Indents</i>	Images are displayed, items are indented as specified by <i>Indents</i>

2.14 CoolBand Object

Purpose: The *CoolBand* object represents an area in a *CoolBar* that contains a child window.

Parents *CoolBar*

Children *Bitmap* *BrowseBox* *Button* *Clipboard* *Combo* *ComboBox* *Cursor* *Edit* *FileBox* *Font* *Grid* *Group* *Icon* *ImageList* *Label* *List* *ListView* *Menu* *Metafile* *MsgBox* *OCXClass*

OLEClient Printer ProgressBar RichEdit Scroll Spinner Static StatusBar SubForm
 TabControl TCPSocket Timer TipField ToolBar ToolControl TrackBar TreeView UpDown

Properties Type Caption Posn Size Visible Event ImageIndex FCol BCol Picture Index
 Data KeepOnClose ChildEdge NewLine GripperMode Dockable UndocksToRoot
 MethodList ChildList EventList PropList

Methods Detach

Events Close Create DockStart DockMove DockRequest DockAccept DockEnd
 DockCancel

Description

The CoolBand object is a container object that represents a band in a *CoolBar*.

A CoolBand can have any combination of a gripper bar, a bitmap, a text label, and a single child object.

A CoolBand may not contain more than one child object, but that child object may itself be a container such as a *ToolControl* or a *SubForm*.

The *Caption* property specifies a text string to be displayed to the left of the CoolBand. The colour of the text is specified by the *FCol* property.

The *ImageIndex* property specifies an optional picture which is to be displayed alongside the *Caption*. If specified, *ImageIndex* is an index into an *ImageList* whose name is referenced via the *ImageListObj* property of the parent *CoolBar*.

The background in a CoolBand may be specified using its *BCol* or *Picture* properties. Although typically, the visible background area is small, it is visible through a *transparent ToolControl*.

The *ChildEdge* property specifies whether or not the CoolBand leaves space above and below its child window.

The *GripperMode* property specifies whether or not the CoolBand has a gripper bar which is used to reposition and resize the CoolBand within its parent *CoolBar*. *GripperMode* may be 'Always' (the default), 'Never' or 'Auto'.

The position of a Cool Band within a *CoolBar* is determined by its *Index* and *NewLine* properties, and by the position and size of preceding CoolBand objects in the same *CoolBar*. For a CoolBand, *Posn* is a read-only property that reports its position but *Posn* may not be used to set it.

The *Index* property specifies the position of a CoolBand within its parent *CoolBar*, relative to other CoolBands and is `IO` dependent. Initially, the value of *Index* is determined by the order in which the CoolBands are created. You may re-order the CoolBands within a *CoolBar*, under program control, by changing *Index* with `WS`.

The *NewLine* property specifies whether or not the CoolBand occupies the same row as an existing CoolBand, or is displayed on a new line within its *CoolBar* parent. The value of *NewLine* in the first CoolBand in a *CoolBar* is always 1, even if you specify it to be 0. You may move a CoolBand to the previous or next row by changing its *NewLine* property (using `WS`) from 1 to 0, or from 0 to 1 respectively.

The 2nd element of the *Size* property determines the width of the CoolBand; the value of the 1st element is read-only.

Size may **only** be specified by `WC`. However, when you create a CoolBand, it will automatically occupy all the available space in the current row, to the right of any preceding CoolBands. Only when you create *another* CoolBand in the *same* row, will the *Size* of the first CoolBand be honoured. The rightmost CoolBand will always extend to the right edge of the *CoolBar*, whatever its *Size*.

If you create two or more CoolBands in the same row and you do not specify *Size*, the first CoolBand will be maximised, and the others minimised.

When the user drags a CoolBand to a different row its *Index* and *NewLine* properties may change, as may the *Index* and *NewLine* properties of any another CoolBand that is affected by the operation.

If you wish to remember the user's chosen layout when your application terminates, you must store the values of *Index*, *Size* and *NewLine* for each of the CoolBands. When your application is next started, you must re-create the CoolBands with the same values of these properties.

2.15 CoolBar Object

Purpose: The CoolBar object acts as a container for CoolBand objects.

Parents ActiveXControl Form

Children CoolBand ImageList Menu Timer

Properties Type Posn Size Align Event ImageListObj FCol BCol CursorObj Data Attach Handle KeepOnClose BandBorders DblClickToggle FixedOrder VariableHeight DockChildren Redraw MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help DockStart DockMove DockRequest DockAccept DockEnd DockCancel

Description

The `CoolBar` and `CoolBand` objects provide an interface to Windows *Rebar Controls*

A `CoolBar` contains one or more bands (`CoolBand` objects). Each band can have any combination of a gripper bar, a bitmap, a text label, and a single child object.

Using the gripper bars, the user may drag bands from one row to another, resize bands in the same row, and maximise or minimise bands in a row.

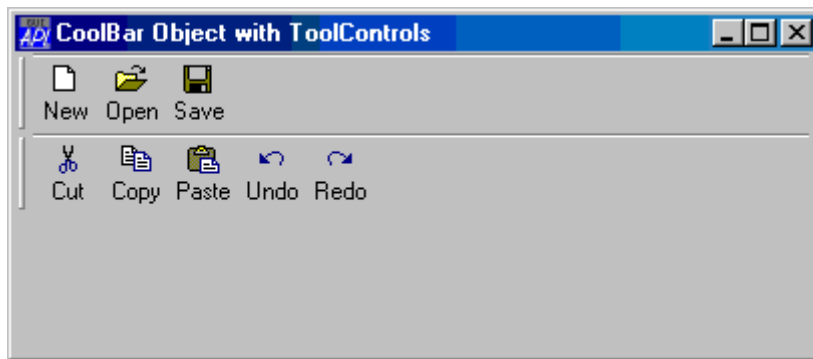
The following example illustrates a `CoolBar` containing 2 `CoolBands` each of which is displaying a `ToolControl` object.

```
'F'□WC'Form' 'CoolBar Object with ToolControls'('Size' 25 50)
'F.IL'□WC'ImageList'('Masked' 0)('MapCols' 1)
'F.IL.'□WC'Bitmap'('ComCtl32' 120)A STD_SMALL

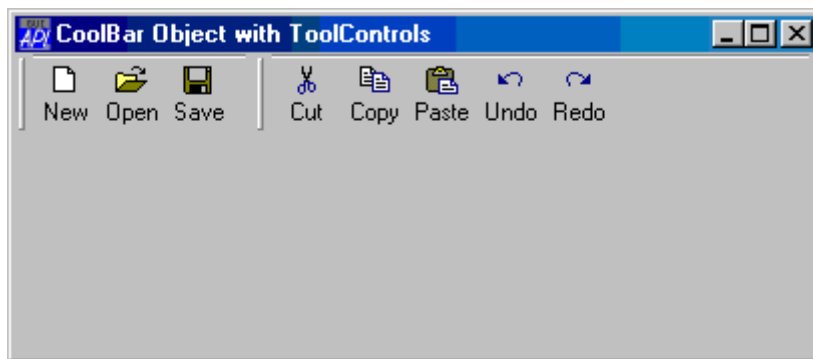
'F.CB'□WC'CoolBar'

:With 'F.CB.C1'□WC'CoolBand'
  'TB'□WC'ToolControl'('ImageListObj' '#.F.IL')
  'TB.B1'□WC'ToolButton' 'New'('ImageIndex' 7)
  'TB.B2'□WC'ToolButton' 'Open'('ImageIndex' 8)
  'TB.B3'□WC'ToolButton' 'Save'('ImageIndex' 9)
:EndWith

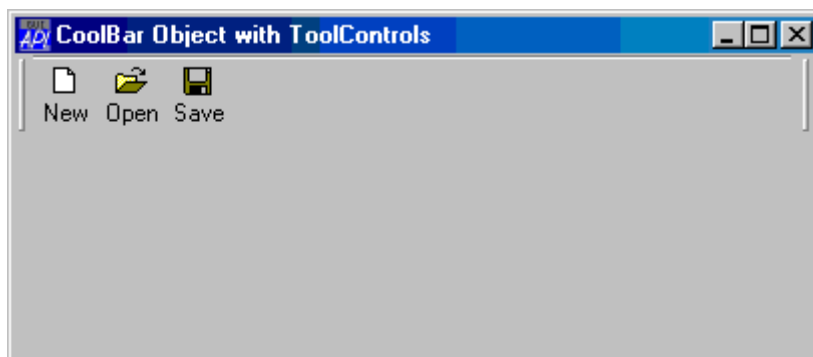
:With 'F.CB.C2'□WC'CoolBand'
  'TB'□WC'ToolControl'('ImageListObj' '#.F.IL')
  'TB.B1'□WC'ToolButton' 'Cut'('ImageIndex' 1)
  'TB.B2'□WC'ToolButton' 'Copy'('ImageIndex' 2)
  'TB.B3'□WC'ToolButton' 'Paste'('ImageIndex' 3)
  'TB.B4'□WC'ToolButton' 'Undo'('ImageIndex' 4)
  'TB.B5'□WC'ToolButton' 'Redo'('ImageIndex' 5)
:EndWith
```

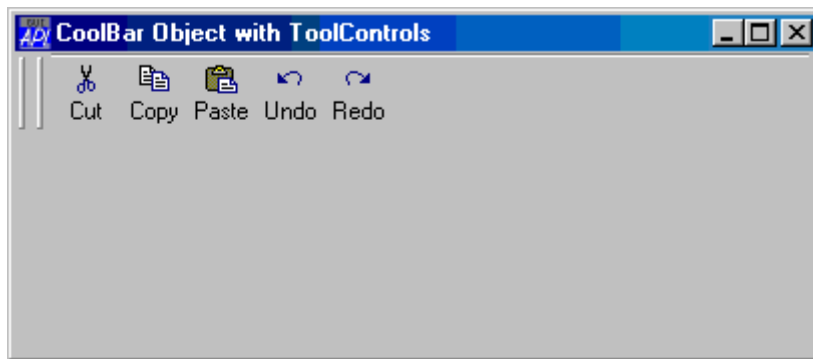
The CoolBar allows the user to organise the CoolBands within it as required. The next three pictures illustrate this feature.



after user has moved band 2 into row 1



after user has maximised band 1



after user has maximised band 2

The second example illustrates a CoolBar containing 3 CoolBands displaying an Edit, Combo and multi-line Edit respectively.

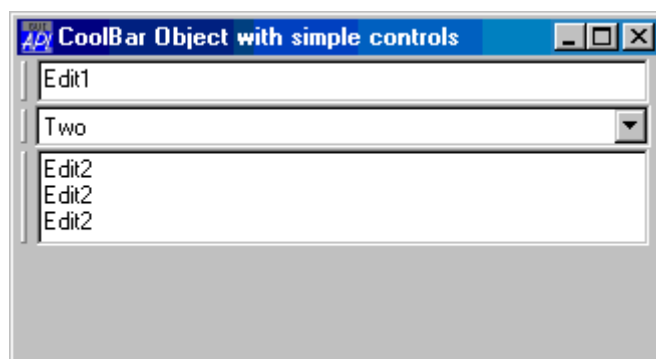
```
'F'[]WC'Form' 'CoolBar Object with simple controls'('Size' 25 40)
'F'[]WS'Coord' 'Pixel'

'F.CB'[]WC'CoolBar'

:With 'F.CB.C1'[]WC'CoolBand'
  'E1'[]WC'Edit' 'Edit1'
:EndWith

:With 'F.CB.C2'[]WC'CoolBand'
  'C1'[]WC'Combo'('One' 'Two' 'Three')('SelItems' 0 1 0)
:EndWith

:With 'F.CB.C3'[]WC'CoolBand'
  'E2'[]WC'Edit'(3 5p'Edit2')('Style' 'Multi')
:EndWith
```



The *VariableHeight* property specifies whether or not the CoolBar displays bands in different rows at the minimum required height (the default), or all the same height.

The *BandBorders* property specifies whether or not narrow lines are drawn to separate adjacent bands. The default is 0 (no lines).

The *DbClickToggle* property specifies whether or not the user must single-click (the default) or double-click to toggle a child *CoolBand* between its maximised and minimised state.

The *FixedOrder* property specifies whether or not the CoolBar displays *CoolBands* in the same order. If *FixedOrder* is 1, the user may move bands to different rows, but the band order is static. The default is 0. Note that when the user moves a *CoolBand* within a CoolBar, its *Index* and (potentially) *NewLine* properties will change to reflect its new position.

If you wish to display pictures in one or more of the *CoolBands* owned by a CoolBar, you do so by setting the *ImageListObj* property to the name of an *ImageList* object which contains the pictures. Pictures are allocated to individual *CoolBands* via their *ImageIndex* properties.

2.16 Cursor Object

Purpose: This object defines a cursor.

Parents ActiveXControl Animation Button Calendar Combo ComboEx CoolBand DateTimePicker Edit Form Grid Group ImageList Label List ListView OLEServer ProgressBar PropertyPage PropertySheet RichEdit Root Scroll SM Static StatusBar SubForm TCPSocket ToolBar ToolControl TrackBar TreeView UpDown

Children Timer

Properties Type File Bits CMap Mask HotSpot KeepBits Event Data Handle Accelerator KeepOnClose MethodList ChildList EventList PropList

Methods Detach FileRead FileWrite

Events Close Create Select

Description

The *File* property defines the name of a cursor file associated with the Cursor object, or it specifies the name of a DLL and the resource number or name of the cursor therein.

If you omit the file extension, the system assumes .CUR. To use an animated cursor you must therefore specify the .ANI extension explicitly.

If the value of the *File* property is set by *Dyalog APL Language: Ws*, no immediate action is taken, but the corresponding file may subsequently be read or written using the *FileRead* or *FileWrite* methods.

The *Bits* and *Mask* properties define the appearance of the cursor. Both are Boolean matrices with a shape of 32 32. The colour of each pixel in the cursor is defined by the following table. Note that a 0 in *Bits* combined with a 1 in *Mask* causes the corresponding pixel to be the colour of the background. This is used to give the cursor a non-rectangular shape.

Bits	0	1	0	1
Mask	0	0	1	1
Pixel	Black	White	Background	Inverse

The *HotSpot* property determines the point within the cursor that registers its position over another object.

A Cursor is **used** by setting the *CursorObj* property of another object to its name or ref.

2.17 DateTimePicker Object

Purpose: The DateTimePicker object is an editable date/time field with an optional drop-down Calendar.

Parents ActiveXControl Form Grid Group PropertyPage SubForm ToolBar

Children Cursor Font Menu MsgBox TCPsocket Timer

Properties Type Posn Size Style Coord Align Border Active Visible Event DateTime MinDate MaxDate CalendarCols Today HasToday CircleToday WeekNumbers MonthDelta HasCheckBox FieldType CustomFormat Sizeable Draggable FontObj CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj IDNToDate DateToIDN

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap DateTimeChange DropDown CloseUp Select

Description

The `DateTimePicker` object represents the built-in Windows date and time picker control. For most purposes, the `DateTimePicker` supersedes the use of `Label`, `Edit` and `Spinner` objects for displaying and entering dates and times. Unlike the `Edit` and `Spinner` objects, it is not possible for the user to enter an invalid date or time into a `DateTimePicker`.

The `Style` property may be either `'Combo'` (the default) or `'UpDown'`. The former provides a drop-down calendar that behaves in the same way as the `Calendar` object and whose appearance and behaviour is controlled by a set of properties namely `CalendarCols`, `CircleToday`, `HasToday`, `MaxDate`, `MinDate`, `MonthDelta`, `Today` and `WeekNumbers` that are common to the `Calendar`. Note that the `Style` property may only be set when the object is created.

If `Style` is `'Combo'`, the `Align` property specifies the horizontal alignment of the drop-down `Calendar` which may be `'Left'` (the default) or `'Right'`.

If `Style` is `'UpDown'`, the `DateTimePicker` includes instead a pair of spinner buttons that allow the user to increment and decrement values in the various sub-fields provided by the control.

The `DateTime` property represents the date and time value that is currently displayed in the object. This is a 4-element vector containing the IDN, hour, minutes and seconds respectively.

The `FieldType` property specifies one of a set of pre-defined date/time formats to be used by the control. This is a character vector that may be empty (the default), `'Date'`, `'DateCentury'`, `'LongDate'`, `'Time'` or `'Custom'`. Specifying an empty vector is the same as specifying `'Date'`. Note that `'DateCentury'` always displays a 4-digit year, regardless of the user's Windows settings.

If `FieldType` is set to `'Custom'`, the format is defined by the `CustomFormat` property. `CustomFormat` is a character vector that may contain a mixture of date/time format elements and body text.

The *HasCheckBox* property is a Boolean value (default 0) that specifies whether or not a checkbox is displayed in the object. This allows the user to specify whether or not the date/time displayed in the *DateTimePicker* is applicable.

2.18 Edit Object

Purpose: Allows user to enter or edit data.

Parents ActiveXControl CoolBand Form Grid Group PropertyPage SubForm ToolBar ToolControl

Children Circle Cursor Ellipse Font Marker Poly Rect Text Timer

Properties Type Text Posn Size Style Coord Align Border Justify Active Visible Event VScroll HScroll SelText Sizeable Dragable FontObj FCol BCol CursorObj AutoConf Data Attach TextSize EdgeStyle Handle Hint HintObj Tip TipObj FieldType MaxLength Decimals Password ValidIfEmpty ReadOnly FormatString Changed Value Translate Accelerator AcceptFiles WantsReturn KeepOnClose Redraw TabIndex Cue ShowCueWhenFocused HasClearButton MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ChooseFont

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap Select BadValue Change KeyError

Description

The value of the *Style* property, which may be 'Single' or 'Multi', determines whether the object presents a single-line data entry field or an area for viewing and editing a large block of text.

Single-Line Edit

The *FieldType* property (which applies only to a single-line Edit object) is either an empty vector (the default) or specifies the type of the field as reflected by its *Value* property. If *FieldType* is empty, the *Value* of the field may be a number or a character vector, according to the contents defined by its *Text* property (this is always a character vector).

The *Align* property, which applies only to single-line Edit objects, specifies the vertical alignment of the text. It may be 'None' (the default), 'Centre' or 'Center'. *Align* may only be set when the object is created and may not subsequently be changed.

If *FieldType* is 'Char' the *Value* of the field is forced to be a character vector, even if the contents defined by its *Text* property is entirely numeric.

If *FieldType* is 'Numeric', 'LongNumeric', 'Currency', 'Date', 'LongDate', or 'Time' the *Value* contains a number. For fields of these types, basic validation is provided during user input. The field is revalidated when the user attempts to "leave" it and at this point the object will generate a *BadValue* event if its contents are inconsistent with its *FieldType*.

Note that when an Edit object is used as the *Input* property of a *Grid*, it is the *Value* of the Edit object (and not its *Text* property) that is used to update the *Values* property (that is, the contents of) the *Grid* when the user moves away from the cell.

The *MaxLength* property defines the maximum number of characters that the user may enter into the object.

The *ValidIfEmpty* property may be 0 (the default) or 1 and specifies whether or not the Edit field is deemed to be valid if it is empty.

The *Password* property specifies the character that is displayed in response to the user typing a character. Normally, *Password* is empty (the default) and the object displays the character that was entered. However, if you set *Password* to (say) an asterisk (*) this symbol will be displayed instead of the characters the user has entered. Note however that the *Text* and *Value* properties will reflect what the user typed.

The *HScroll* property determines whether or not the data may be scrolled. If *HScroll* is 0, the data is not scrollable, and the user cannot enter more characters once the field is full. If *HScroll* is -1 or -2 the field is scrollable, and there is no limit on the number of characters that can be entered. In neither case however is a horizontal scrollbar provided. *HScroll* may only be set when the object is created and may not subsequently be changed.

Multi-Line Edit

If the *Style* is 'Multi', *Text* may set using a simple character vector, a simple matrix, or a vector of vectors. If *Text* is specified by a matrix or by a vector of vectors, "new-line" characters are automatically added at the end of each line in the Edit control.

The user may insert a "new-line" character in the text by pressing Ctrl-Enter. If *Text* was set by a matrix, it is returned as a matrix. Otherwise it is returned as a vector of vectors. "New-line" characters are not returned. If *Text* was not specified by *Dyalog APL Language: Wc* or *Dyalog APL Language: Ws* it is returned an empty matrix (1 0p ' '). However, if *Text* was not specified, but the user types and then empties the field, it is returned as an empty nested array (, c ' ')

The *Justify* property determines whether the text in a multi-line Edit object is 'Left', 'Right', or 'Centre' justified. Setting *Justify* to 'Centre' or 'Right' also forces word-wrapping and disables horizontal scrolling, whatever the value of *HScroll*. Note that the keyword 'Centre' may also be spelled 'Center'. *Justify* may only be specified when the object is created using `WC`.

If *Justify* is 'Left', the *HScroll* property determines whether or not text may be scrolled horizontally. If *HScroll* is set to -2, each individual line may be any length, but the object does not have a horizontal scrollbar. Sideways scrolling is achieved using the cursor keys, or by typing. If *HScroll* is -1, each individual line may be of any length and the object will have a horizontal scrollbar. If *HScroll* is 0, lines are automatically "word-wrapped" at the right edge of the object. This means that the number of lines displayed may be greater than the number of lines implied by the rows of the matrix or the number of vectors supplied. In particular, if you specify a single long vector, it will be broken up into lines for you on the display, but still returned as a single vector by *Dyalog APL Language: Wg*.

The *VScroll* property determines whether or not data may be scrolled vertically and whether or not the object has a vertical scrollbar. A value of 0 inhibits scrolling; -2 means scrollable, without a scrollbar; -1 means scrollable with a scrollbar. *VScroll* may only be set when the object is created and may not subsequently be changed.

The setting of *Justify* forces word-wrapping.

The *SelText* property identifies the portion of the text that is selected. It may be used to pre-select (and highlight) a part of the text, or to report the part of the text selected by the user. *SelText* is a 2-element integer vector which specifies the start and end of the selected area. Its structure depends upon the nature of the data specified by *Text*. See the description of *SelText* for details.

If the user changes any data in the field **and** attempts to change focus to another object, the Edit object will generate a *Change* event. You can use this to validate the new data in the field.

Note

For full functionality (in particular, for the *Cue* property to apply), the Edit object requires that *Native Look and Feel*

is enabled.

2.19 Ellipse Object

Purpose: A Graphical object to draw ellipses, arcs, and pie-slices.

Parents ActiveXControl Animation Bitmap Button ButtonEdit Combo ComboEx Edit Form Grid Group Label List ListView MDIClient Metafile Printer ProgressBar PropertyPage PropertySheet RichEdit Scroll Spinner Static StatusBar SubForm TabBar TipField ToolBar TrackBar TreeView UpDown

Children Timer

Properties Type Points Size FCol BCol Start End ArcMode LStyle LWidth FStyle FillCol Coord Visible Event Dragable OnTop CursorObj AutoConf Data Accelerator KeepOnClose DrawMode MethodList ChildList EventList PropList

Methods Detach

Events Close Create DragDrop MouseDown MouseUp MouseMove MouseDbClick Help Select

Description

This object duplicates much of the functionality of the *Circle* object, but differs in two major respects. Firstly, ellipses, circles, and arcs are specified in terms of their **bounding rectangles**, rather than in terms of their centre(s) and radii. Secondly, the Ellipse object behaves like any other (rectangular) object when it is resized by its parent. The *Circle* object behaves differently in that when resized by its parent, it maintains a constant ratio between its **physical** height and width.

The *Points* property specifies one or more sets of co-ordinates which define the position(s) of one or more bounding rectangles. The position is defined to be the position of the corner that is **nearest** to the origin of its parent. The default is therefore its top-left corner.

The *Size* property specifies the height and width of each bounding rectangle, measuring away from the origin. To obtain a perfect circle, you must take the **aspect ratio** of the

device into account. This is available from the *DevCaps* property of the *Root* and *Printer* objects. Alternatively you can use the *Circle* object.

The *Start* and/or *End* properties are used to draw partial ellipses and circles. They specify start and end angles respectively, measuring from the x-axis at the centre of the bounding rectangle in a counter-clockwise direction and are expressed in radians. The type of arc is controlled by *ArcMode* as follows:

ArcMode	Effect
0	An arc is drawn from <i>Start</i> to <i>End</i> .
1	An arc is drawn from <i>Start</i> to <i>End</i> . In addition, a single straight line is drawn from one end of the arc to the other, resulting in a segment.
2	An arc is drawn from <i>Start</i> to <i>End</i> . In addition, two lines are drawn from each end of the arc to the centre, resulting in a pie-slice.

LStyle and *LWidth* define the style and width of the lines used to draw the boundaries of the ellipse(s), circle(s) or arc(s). *FCol* and *BCol* determine the colour of the lines.

FStyle specifies whether or not the ellipse(s), circle(s) or arc(s) are filled, and if so, how. For a solid fill (*FStyle* 0), *FillCol* defines the fill colour used. For a pattern fill (*FStyle* 1-6) *FillCol* defines the colour of the hatch lines and *BCol* the colour of the spaces between them.

The value of *Dragable* determines whether or not the object can be dragged. The value of *AutoConf* determines whether or not the Ellipse object is resized when its parent is resized.

The structure of the property values is best considered separately for single and multiple ellipses, circles or arcs.

Single Ellipse, Circle or Arc

For a single ellipse, circle or arc, *Points* is a 2-element vector which specifies the y-coordinate and x-coordinate of the top-left corner of the bounding rectangle.

Size is also a simple 2-element vector whose elements specify the height and width of the bounding rectangle.

LStyle and *LWidth* are both simple scalar numbers.

FStyle is either a single number specifying a standard fill pattern, or the name of a *Bitmap* object which is to be used to fill the ellipse, circle or arc.

FCol, *BCol* and *FillCol* are each either single numbers representing standard colours, or 3-element vectors which specify colours explicitly in terms of their RGB values.

EXAMPLES

First make a *Form* :

```
'F' WC 'Form'
```

Draw a complete ellipse within the bounding rectangle located at (y=10, x=5) with (height=30, width=50) :

```
'F.E1' WC 'Ellipse' (10 5)(30 50)
```

Draw an elliptical arc within the same bounding rectangle as above, occupying the upper right quadrant (0 to 90 degrees):

```
'F.E1' WC 'Ellipse' (10 5)(30 50)('End'(0.5))
```

Ditto, but between 45 and 135 degrees :

```
'F.E1' WC 'Ellipse' (10 5)(30 50)('Start'(0.25))('End'(0.75))
```

Ditto, but join the points of the arc to the centre of the ellipse, making a "pie-slice":

```
'F.E1' WC 'Ellipse' (10 5)(30 50)('Start'(0.25))('End'(0.75))  
( 'ArcMode' 2)
```

Ditto, but use a green line and solid red fill :

```
'F.E1' WC 'Ellipse' (10 5)(30 50)('Start'(0.25))('End'(0.75))  
( 'ArcMode' 2)('FCol' 0 0 255)('FStyle' 0)('FillCol' 255 0 0)
```

Multiple Ellipses, Circles or Arcs

To draw a set of ellipses, circles, or arcs with a single name, *Points* may be a simple 2-element vector (specifying the location of all the bounding rectangles), **or** a 2-column matrix whose first column specifies their y-coordinates and whose second column specifies their x-coordinates, **or** a 2-element nested vector whose first element specifies their y-coordinate(s) and whose second element specifies their x-coordinate(s).

Likewise, *Size* may be a simple 2-element vector (applying to all the bounding rectangles), **or** a 2-column matrix whose first column specifies their heights and whose

second column specifies their widths, **or** a 2-element nested vector whose first element specifies their height(s) and whose second element specifies their width(s).

If specified, *Start* and/or *End* define arcs in terms of the angles made by drawing a line from the centre of the bounding box to the two ends of the arc. Both properties may be simple scalars, or vectors containing one element per arc drawn.

If *Start* is specified, but not *End*, end angles default to $(-1 \div + \backslash \text{Start}) \cdot \pi / 2$. If *End* is specified, but not *Start*, start angles default to $0, -1 \div + \backslash \text{End}$

This means that you can draw a pie-chart using either *Start* or *End* angles; you do not have to specify both.

ArcMode, *LStyle* and *LWidth* may each be simple scalar values (applying to all the ellipses, circles or arcs) or simple vectors whose elements refer to each of the corresponding ellipses, circles or arcs in turn.

FStyle may be a simple scalar numeric or a simple character vector (*Bitmap* name) applying to all rectangles, or a vector whose elements refer to each of the corresponding ellipses, circles or arcs in turn.

Similarly, *FCol*, *BCol* and *FillCol* may each be single numbers or a single (enclosed) 3-element vector applying to all the rectangles. Alternatively, these properties may contain vectors whose elements refer to each of the rectangles in turn. If so, their elements may be single numbers or nested RGB triplets, or a combination of the two.

The *Coord*, *Dragable* and *Data* properties are specified for the object as a whole, and may not be allocated different values for each individual ellipse, circle or arc that is drawn.

EXAMPLES

First make a *Form* :

```
'F' □WC 'Form'
```

Draw two ellipses in bounding rectangles located at (y=5, x=10) and (y=5, x=60), each of (height=40, width=10)

```
'F.E1' □WC 'Ellipse' ((5 5)(10 60)) (40 10)
```

Ditto, using scalar extension for (y=5) :

```
'F.E1' □WC 'Ellipse' (5(10 60)) (40 10)
```

Ditto, but draw the first with (height=40, width=30) and the second with (height=20, width=10) :

```
'F.E1' WC 'Ellipse' (5(10 60)) ((40 20)(30 10))
```

Draw an elliptical Pie-Chart in a bounding rectangle located at (y=5, x=10) with a height and width equal to 40% of the height and width of the parent *Form*. Each of the 4 pie-slices is bounded by a black line :

```
Data ←12 27 21 40
ANGLES←0, -1↓((○2)÷+/Data)×+\Data
COLS←(255 0 0)(0 255 0)(255 255 0)(0 0 255)
PATS←1 2 3 4

'F.PIE' WC 'Ellipse'(5 10)(40 40)
      ('Start' ANGLES)('ArcMode' 2)
      ('FCol' (c0 0 0))('FStyle' PATS)
      ('FillCol' COLS)
```

2.20 FileBox Object

Purpose: Prompts user to select a file.

Parents ActiveXControl CoolBand Form Grid OLEServer PropertyPage PropertySheet Root StatusBar SubForm TCPSocket ToolBar ToolControl

Children Timer

Properties Type Caption Directory Filters File FileMode Style Event Index Data EdgeStyle KeepOnClose MethodList ChildList EventList PropList

Methods Detach Wait

Events Close Create FileBoxOK FileBoxCancel Select

Description

The FileBox object implements the standard Windows FileSelection Dialog Box. This is a "modal" object. When you create a FileBox with *Dyalog APL Language: Wc*, it is initially invisible and the user cannot interact with it. To use it, you must execute *Dyalog APL Language: Dq* with the name of the FileBox as its right argument. This causes the FileBox to be displayed. During the "local" *Dyalog APL Language: Dq* the user may interact **only** with the FileBox, or with other applications. When the user terminates

the operation (by pressing the "Save", "Open", or "Cancel" Buttons, or by closing the window) the "local" *Dyalog APL Language: Dq* terminates, and the FileBox disappears.

When the "local" *Dyalog APL Language: Dq* is terminated, the FileBox generates either an *FileBoxOK*(71) or *FileBoxCancel*(72) event. The former is generated when the user presses the "Save" or "Open" button; the latter when the user presses the "Cancel" button or closes the FileBox.

The *FileMode* property is a character vector which indicates the mode in which the selected file is going to be opened. *FileMode* may be 'Read' (the default) or 'Write'. If *FileMode* is 'Write', files listed in the File Selection Box are "greyed", although they may still be selected.

The *Caption* property determines the text that appears in the title bar of the FileBox window. If undefined, *Caption* defaults to "Save As" if *FileMode* is 'Write' or to "Open" if *FileMode* is 'Read'. Similarly, if *FileMode* is 'Write', the button captions are "Save" and "Cancel", and if *FileMode* is 'Read' the button caption are "Open" and "Cancel".

The *Directory* property contains a simple character vector which specifies the initial directory from which a list of suitable files is displayed.

```
'F' □WC 'FileBox' 'The FileBox Object' 'C:\WDYALOG'
□DQ 'F'
```

The *Style* property specifies whether the user may choose a single file name ('Single' which is the default) or several file names ('Multi').

The *Filters* property is a nested scalar or vector containing a list of filters. Each filter is a 2-element vector of character vectors which contain a file type mask and a file type description respectively. The file type descriptions appear in a drop-down combo box labelled "List File of Type". When the user selects one of these, the currently selected directory is searched for files which match the corresponding mask. The default value of *Filters* is an empty vector. This gives a file type mask of "." and a file type description of "All Files (.)". Hence an empty vector is equivalent to (c'*. *' 'All Files (*.*)').

The *Index* property determines which of the filters is initially selected. Its default value is □IO.

Note that when *Dyalog APL Language: Dq* terminates with *FileBoxOK*, the *File*, *Directory*, and *Index* properties are updated to reflect the contents of the fields within the FileBox.

The operating system imposes limits on both the length of the name of the file, and on the total path length. In version 20.0 attempting to set the `File` or `Directory` Properties to too long a name will generate a `DOMAIN ERROR`, while attempting to use too long a File name within the `FileBox` will result in the appearance of an error `MessageBox`.

2.21 Font Object

Purpose: Loads a font resource

Parents `ActiveXControl` `Animation` `Bitmap` `Button` `ButtonEdit` `Calendar` `Combo` `ComboEx` `CoolBand` `DateTimePicker` `Edit` `Form` `Grid` `Group` `Label` `List` `ListView` `MDIClient` `Metafile` `OLEServer` `Printer` `ProgressBar` `PropertyPage` `PropertySheet` `RichEdit` `Root` `Scroll` `Spinner` `Static` `StatusBar` `SubForm` `TabBar` `TCPsocket` `TipField` `ToolBar` `ToolControl` `TrackBar` `TreeView` `UpDown`

Children `Timer`

Properties `Type` `PName` `Size` `Fixed` `Italic` `Underline` `Weight` `Rotate` `CharSet` `Coord` `Data` `Handle` `KeepOnClose` `MethodList` `ChildList` `EventList` `PropList`

Methods `Detach` `ChooseFont`

Events `Close` `Create` `FontOK` `FontCancel` `Select`

Description

This object loads a Windows font into memory ready for use by another object.

The characteristics of the font are specified by its properties as follows :

<i>PName</i>	A character vector containing the name of the font face. The default is 'System'. Note that case is ignored when you specify the name, although it will be returned correctly by <i>Dyalog APL Language: Wg</i> .
<i>Size</i>	An integer that specifies the character height of the font in pixels.
<i>Fixed</i>	A Boolean value that specifies whether the font is fixed-width (1) or proportional (0).
<i>Italic</i>	A Boolean value that specifies whether the font is italicised (1) or not (0).
<i>Underline</i>	A Boolean value that specifies whether the font is underlined (1) or not (0).
<i>Weight</i>	An integer in the range 0-1000 that specifies how bold or heavy the font is (1000 = most bold).
<i>Rotate</i>	A numeric scalar that specifies the angle of rotation of the font in radians. The angle is measure from the x-axis in a counter-clockwise direction.
<i>CharSet</i>	An integer that specifies the character encoding.

The *Coord* property may be set to 'Pixel', 'ScaledPixel' or 'RealPixel' when the object is created, but *Coord* may not subsequently be changed.

If *Coord* is 'Pixel', it is interpreted as either 'RealPixel' or 'ScaledPixel' according to the value of the **Dyalog_Pixel_Type** parameter, which is either ScaledPixel or RealPixel. See *Installation/Configuration: Dyalog Pixel Type*.

If this parameter is not specified, the default is RealPixel. So by default, when you set Coord to Pixel, it will be treated as RealPixel.

If you are using 'ScaledPixel', this means that your fonts will also be scaled up automatically, as well as the sizes of the controls in which they are used.

When you ask Windows to allocate a font, you may specify as many or as few of these properties as you wish. Windows actually supplies the font that most closely matches the attributes you have specified. The matching rules it uses are complex, and may be found in the appropriate Windows documentation.

The values of the above properties after *Dyalog APL Language: Wc* or *Dyalog APL Language: Ws* reflect the attributes of the font which has been allocated by Windows, and not necessarily the values you have specified. Furthermore, it is possible that changing the value of one property will cause the values of others to be changed.

2.22 Form Object

Purpose: This is a top-level window used to contain other objects (controls).

Parents ActiveXControl Form OLEClient OLEServer Root SubForm TCPSocket

Children ActiveXControl Animation Bitmap BrowseBox Button ButtonEdit Calendar Circle Clipboard ColorButton Combo ComboEx CoolBar Cursor DateTimePicker Edit Ellipse FileBox Font Form Grid Group HTMLRenderer Icon Image ImageList Label List ListView Locator Marker MDIClient Menu MenuBar Metafile MsgBox OCXClass OLEClient OLEServer Poly Printer ProgressBar PropertySheet Rect RichEdit Scroll SM Spinner Splitter Static StatusBar SubForm SysTrayItem TabBar TabControl TCPSocket Text Timer TipField Toolbar ToolControl TrackBar TreeView UpDown

Properties Type Caption Posn Size Coord State Border Active Visible Event Thumb Range Step VScroll HScroll Sizeable Moveable SysMenu MaxButton MinButton HelpButton OKButton SIPMode SIPResize FontObj BCol Picture OnTop IconObj CursorObj AutoConf YRange XRange Data TextSize EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles KeepOnClose Dockable Docked DockShowCaption DockChildren UndocksToRoot MaskCol AlphaBlend Redraw TabIndex PageSize MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ChooseFont Wait

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDblClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap StateChange DockStart DockMove DockRequest DockAccept DockEnd DockCancel Select FrameContextMenu DyalogCustomMessage1 FontOK FontCancel VScroll HScroll VThumbDrag HThumbDrag

Description

The *Posn* property specifies the location of the **internal** top-left corner of the window relative to the top-left corner of the screen. If the window has a title bar and/or border, you must allow sufficient space for them. Similarly, the *Size* property specifies the internal size of the window excluding the title bar and border. The default for *Size* is 50% of the screen height and width. The default for *Posn* places the Form in the middle of the screen.

Normally, a Form has a title bar, a system menu box, a border and maximise and minimise buttons. To disable the System Menu box, set *SysMenu* to 0. To disable one or both of the maximise/minimise buttons, set *MaxButton* and/or *MinButton* to 0.

The *HelpButton* property specifies that a Question (?) button appears in the title bar of the Form. However, this does not apply if the Form has a maximise or minimise button which both take precedence. The user may obtain help by clicking on the Question (?) button and then on a control in the Form. It is up to you to provide the help by responding to the *Help* event on the control.

By default, a Form may be moved and resized using the mouse. These actions are achieved by dragging on the title bar and border respectively. It follows that a Form that is *Moveable* **must** have a title bar, and one that is *Sizeable* **must** have a border, regardless of the value of other properties. Also, if you specify any of *SysMenu*, *MaxButton* or *MinButton*, the window must have a title bar in which to place these controls. A title bar itself requires a border. To obtain a window without a title bar, you must therefore set *Moveable*, *SysMenu*, *MaxButton* and *MinButton* to 0. Note that setting *Caption* does **not** force a title bar on the window.

If *Sizeable* is 1, the window will have a double-line border, regardless of the values of other properties. If *Sizeable* is 0, and any one or more of *Moveable*, *SysMenu*, *MaxButton*, *MinButton* or *Border* is 1, the window will have a 1-pixel border. Only if all these properties are 0 will the window be borderless. To obtain a dialog box that may only be moved or closed, set *Border* to 2.

Note that the default value for *Caption* is an empty character vector which results in a blank title.

To obtain a standard dialog box with 3-dimensional appearance, create a Form with *Border* set to 2 and *EdgeStyle* set to 'Dialog', for example:

```
'F' □WC 'Form' 'Dialog Box' ('EdgeStyle' 'Dialog') ('Border' 2)
```

The *State* property has the value 0 if the window is currently displayed in its "normal" state, 1 if it is currently displayed as an icon, and 2 if it is currently maximised and displayed full-screen. This property does not just report the current state, but can be used to set the state under program control.

The *VScroll* and *HScroll* properties determine whether or not a Form has a vertical and horizontal scrollbar respectively. These properties are set to ~1 to obtain a scrollbar. Their default value is 0 (no scrollbar). *VScroll* and *HScroll* may only be set when the object is created and may not subsequently be changed. The *Range* property is a 2-element vector that specifies the maximum value for the vertical and horizontal scrollbars respectively. The *PageSize* property is a 2-element vector that specifies the

sizes of the thumbs in each scrollbar. The *Step* property is a 4-element vector that specifies the sizes of the small and large change. Its first two elements refer to the vertical scrollbar, elements 3 and 4 refer to the horizontal scrollbar. The *Thumb* property is a 2-element vector that both reports and sets the position of the thumb in the vertical and horizontal scrollbars respectively. When the user attempts to move the thumb in one of the scrollbars, the Form generates a *VScroll* or *HScroll* event.

VScroll and *HScroll* cannot be changed using *Dyalog APL Language: Ws*. However, you can make a scrollbar disappear by setting the corresponding element of *Range* to 1, thus allowing you to dynamically switch the scrollbar off and on. Note however that doing so will change the size of the Form.

Setting the *FontObj* property on a Form does not affect the text in its title bar. However, the value of *FontObj* will (unless over-ridden) be inherited by all of the objects within the Form.

The background of the Form may be coloured using *BCol*. The default value for *BCol* is the Windows Button Face colour unless *EdgeStyle* is set to 'None' or 'Default' in which case it is the Windows background colour. Alternatively, the background of a Form can be defined using a *Bitmap* or *Metafile* object whose name is defined by the *Picture* property. A *Metafile* is automatically scaled to fit the Form. A *Bitmap* can be tiled or scaled. See *Picture* property for details.

The *OnTop* property is either 0 or 1. If it is 0, the Form assumes its normal position within the stack of windows on the screen and is only brought to the front when it receives the input focus. If *OnTop* is set to 1, the Form is always brought to the front even when it doesn't have the focus. If more than one Form has *OnTop* set to 1, the stacking order of this set of Forms is defined by the order in which they were created.

A Form can be created as a child of another Form. If so, it has the following characteristics:

- A child Form always appears on top of its parent Form (although it is not constrained by it)
- When you minimize a parent Form, its child Forms disappear.
- Making the parent Form invisible or inactive has no effect on a Child Form.

Note that the *Posn* and *Size* properties of a child Form are expressed in screen coordinates and are not given relative to its parent.

2.23 Grid Object

Purpose: Spreadsheet object for displaying and editing data.

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm

Children Bitmap BrowseBox Button Circle ColorButton Combo Cursor DateTimePicker Edit Ellipse FileBox Font Icon Image Label Marker Menu MsgBox OCXClass Poly Rect Spinner Text Timer TrackBar

Properties Type Values Posn Size FCol BCol Coord Border Active Visible Event VScroll HScroll SellItems Sizeable Dragable FontObj CursorObj AutoConf Index YRange XRange Data Attach TextSize EdgeStyle Handle Hint HintObj Tip TipObj FormatString RowTitles ColTitles CurCell TitleWidth CellHeights CellWidths TitleHeight CellFonts Input CellTypes AutoExpand CellSelect ResizeRows ResizeCols ResizeRowTitles ResizeColTitles ClipCells InputModeKey InputMode GridFCol GridBCol ShowInput CellSet RowTitleFCol ColTitleFCol RowTitleDepth ColTitleDepth RowTitleAlign ColTitleAlign OverflowChar AlignChar GridLineFCol GridLineWidth RowLineTypes ColLineTypes EnterReadOnlyCells RowTitleBCol ColTitleBCol RowTreeDepth RowTreeStyle RowTreeImages ColSortImages SelectionColor SelectionColorAlpha SelectionBorderWidth HighlightHeaders Translate Accelerator AcceptFiles KeepOnClose Redraw InputProperties TabIndex AlwaysShowSelection AlwaysShowBorder RowHiddenDepth CornerTitleBCol MethodList ChildList EventList PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj DelRow DelCol SetCellType RowChange ColChange Undo SetCellSet RowSetVisibleDepth ColSorted DuplicateRow DuplicateColumn CellFromPoint GetCellRect LockRows LockColumns AddComment DelComment GetComment

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseEnter MouseLeave GridKeyPress CellChange CellMove AddRow AddCol CellError CellOver CellDown CellUp CellDbClick CellChanged GridSelect Expanding Retracting SetRowSize SetColSize GridCut GridCopy GridPaste GridDelete GridPasteError GridDropSel GridCopyError IndexChanged ShowComment HideComment ClickComment Select

Description

The *Values* property is a matrix whose elements are displayed in the cells of the Grid. An element (and therefore a cell) may contain a single number, a single character, a character vector or a character matrix.

The *CellHeights* property specifies the height of each of the rows of the spreadsheet. It may be a single value which applies to all rows, or a vector with one element per row. The *CellWidths* property determines the width of each column of the spreadsheet. It too may be a single value or a vector with one element per column.

The *RowTitles* property is either an empty character vector (the default) or a vector of character vectors that specify row titles displayed to the left of the cells in the Grid. If *RowTitles* is not specified, the Grid labels each row with its row number. The *ColTitles* property is similar and is used to specify column headings. If *ColTitles* is not specified, the Grid displays "standard" spreadsheet column headings A-Z, then AA-AZ and so forth.

The *TitleHeight* property specifies height of the column headers. If this is set to 0, the column titles will not be displayed. Similarly, the *TitleWidth* property specifies the width of the row titles and again a value of zero disables the row titles.

The *CornerTitleBCol* property specifies the background colour of the rectangle in the top-left corner of the Grid.

The *FontObj* property may be used to specify the font to be used for the Grid as a whole, including the titles. The *CellFonts* property may be used to specify fonts for individual cells.

The *FCol* and *BCol* properties may each specify a single colour for the Grid as a whole, or may specify a vector of colours whose elements are mapped to individual cells through the *CellTypes* property.

The *CellFonts* property is either a character vector or a vector of character vectors that specifies the name of a single font object to be used for all cells in the Grid, or a vector of character vectors that specifies a set of font objects that are mapped to individual cells through the *CellTypes* property.

The *Input* property is a character vector that specifies the name of an object which is to be associated with every cell in the Grid, or a vector of names whose elements are mapped to individual cells through the *CellTypes* property. These objects may be of type *Button*, *ColorButton*, *Combo*, *Edit*, *Label*, *Spinner* or *TrackBar*. In addition, the *Input* property may specify instances of *OCXClass* objects (ActiveX controls) and *NetType* objects (.NET classes).

If the *Input* property is empty (the default) the user may browse the data in the spreadsheet but may not alter it. Furthermore, no feedback is provided as to which is the current cell. If the *Input* property specifies the name of an object that is the child of the Grid itself, this object *floats* from cell to cell as the user moves around the spreadsheet, and the current cell is identified by its presence.

If the *Input* property specifies the name of an external object (that is, an object that is *not* a child of the Grid), the contents of the current cell are copied into that object as the user moves around the spreadsheet. In addition, the current cell is identified by a

thick border. In either case, the associated object is used to impose formatting and validation.

If the *Input* property specifies the name of a *Label* object, that object is used to impose formatting, but the data is protected and may not be changed. If the *Label* is a child of the Grid, it moves from cell to cell, and its characteristics (*Border*, *FCol*, *BCol* and *FontObj*) can be used to identify the current cell. If the *Label* is an external one, no visual feedback is provided; even though the current cell (reflected by the *CurCell* property) changes as the user moves around the Grid.

If the *Input* property specifies one or more instances of OCXClass objects (ActiveX controls) and *NetType* objects (.NET classes), the *InputProperties* property is used to map the *Values* property of the Grid to specific properties of the external object.

The *CellTypes* property is either an empty numeric matrix (the default) or an integer matrix of the same shape as *Values*. If specified, each element of *CellTypes* determines the index into various properties, including the *FCol*, *BCol*, *CellFonts* and *Input* properties, to be used for the corresponding cell. For example, if an element in *CellTypes* is 3, the 3rd element of *FCol* is used for the foreground colour of the corresponding cell, the 3rd element of *BCol* specifies the background colour, and so forth.

The *CurCell* property may be used to set or query the current cell. The current cell is the cell which the user has picked by clicking the mouse over it or by using the cursor keys. *CurCell* is a 2-element vector containing the current cell's row number and column number respectively and is $\square IO$ dependent. The *Index* property specifies the row and column number of the cell in the top-left corner of the Grid. It too is $\square IO$ dependent.

The *AutoExpand* property is a 2-element Boolean vector which specifies whether (1) or not (0) new rows and columns are added when the user presses the corresponding cursor key when at the end of the block of cells. Its default value is (0 0).

The Grid object reports a *CellDown* event when the user depresses a mouse button over a cell. The event message contains the row and column address of the cell in question which is $\square IO$ dependent. It also reports a similar *CellUp* event when the mouse button is released and a *CellDbClick* event when it is double-clicked. The number of the mouse button and the state of the shift keys are also reported.

When the user moves to another cell, the Grid object reports a *CellMove* event. This simply reports the address of the new cell and may be used to take some appropriate action when a particular cell is picked. If the user alters the data in a cell and then

attempts to move to another, the Grid reports a *CellChange* event. This can be used to perform validation.

Alternatively, the Grid may report a *CellChanged* event which occurs after the *Values* property has been updated with the new cell contents. This may be used to perform immediate recalculation.

The *AddRow* event is generated if the current cell is in the last row of the Grid and the user presses Cursor Down. By default, this operation adds a new row to the Grid, but you can attach a callback to the *AddRow* and selectively disable this default action if required. The *AddCol* event works in a similar manner for columns. Although the user has no direct means of *inserting* a row or column, your application can do this by calling *AddRow* or *AddCol* as a method on the Grid object. Typically this would be done in response to the user selecting a *MenuItem* or pressing a *Button*.

The *ColChange*, *RowChange*, *DelRow*, *DelCol* and *Undo* methods allow your application to perform these corresponding operations.

The Grid object maintains a buffer of the most recent 8 changes made by the user since the *Values* property was last set by *Dyalog APL Language: Wc* or *Dyalog APL Language: Ws*. Your application can restore these changes one by one by calling the *Undo* method. The *Undo* method restores the most recent change made by the user and removes that change from the undo stack. It is therefore not possible to "undo an undo".

The Grid supports the selection of a block or blocks of cells using the mouse and/or the keyboard. The ability to select a range of cells is determined by the *CellSelect* property. When the user performs a selection, the Grid generates a *GridSelect* event. The range of cells currently selected is given by the *SelItems* property.

If a block of cells has been selected, the user may delete the contents, and cut or copy the contents of the cells to the clipboard by pressing Delete, Shift+Delete or Ctrl+Insert respectively. These operations also generate *GridDelete*, *GridCut* and *GridCopy* events which you can selectively disable using a callback function. You can also perform these operations under program control by calling them as methods.

If more than one block of cells has been selected, these operations are honoured only if the blocks begin and end on the same rows or begin and end on the same columns. If so, the data placed in the clipboard is the result of joining the blocks horizontally or vertically as appropriate.

The user may paste data from the clipboard into a Grid by pressing Shift+Insert. Data is pasted into the currently selected block of cells, or, if there is no selection, data is pasted starting at the current cell (*CurCell*). The operation also generates a *GridPaste* event, and, if the operation cannot proceed, a *GridPasteError* event.

If you move the mouse pointer over any of the four edges of a selected block of cells, the cursor changes to an arrow. You may now click and drag the border of the selected cells with the mouse.

If you press the Ctrl key at the same time, the contents of the selected cells are *copied* to the new location, replacing the values in the block of cells onto which they are dropped. Otherwise, the operation is treated as a *move* and the original block of cells is emptied. This operation also generates a *GridDropSel* event. You may only move or copy a *single* block of cells in this way.

The user may be permitted to resize the rows and/or columns of a Grid. This is controlled by the *ResizeRows* and *ResizeCols* properties whose default values are 0. To allow the user to resize, set either or both to 1. You can also specify a Boolean vector to allow specific rows/columns to be resized while others are fixed. Two additional properties named *ResizeRowTitles* and *ResizeColTitles* determine whether or not the user may alter the width of the row titles and the height of the column titles.

If resizable, the cursor changes to a double-heads arrow when the user moves the mouse pointer over the lines between the row and/or column titles. The user may click and drag with the mouse to the desired size. The user may also double-click. This causes the row or column to be resized to fit the data. Both operations generate a *SetColSize*, or *SetRowSize* event.

When you edit data in a Grid, the editing behaviour and the action of the cursor movement keys is determined by the *InputMode* and *InputModeKey* properties.

The *GridFCol* property specifies the colour of all the grid lines. Alternatively, the *GridLineFCol*, *GridLineWidth*, *RowLineTypes* and *ColLineTypes* properties may specify the appearance for individual grid lines.

The *GridBCol* property specifies the colour used to fill the area between the end of the last column of data and the right edge of the Grid and between the bottom row of data and the bottom edge of the Grid.

The *RowTitleFCol* and *ColTitleFCol* properties specify the colours to be used for the row and column titles respectively.

The *ClipCells* property determines whether or not the Grid displays partial cells. The default is 1. If you set *ClipCells* to 0, the Grid displays only complete cells and automatically fills the space between the last visible cell and the edge of the Grid with the *GridBCol* colour.

The *CellSet* property is a Boolean array that marks which cells are *set* (that is, have values) and which are empty. This allows you to edit large numeric matrices which contain empty cells without a severe workspace penalty.

The *HScroll* and *VScroll* properties specify whether or not horizontal and vertical scrollbars are displayed. Either property may be given the value `-3` which forces the corresponding scrollbar to appear always. *VScroll* and *HScroll* may only be set when the object is created and may not subsequently be changed.

The Grid object supports *comments* in a manner that is consistent with the way that comments are handled by Microsoft Excel. If a comment is associated with a cell, a small red triangle is displayed in its top right corner. When the user rests the mouse pointer over a commented cell, the comment is displayed as a pop-up with an arrow pointing back to the cell to which it refers. The comment disappears when the mouse pointer is moved away. This is referred to as *tip* behaviour. Comments may also be associated with row and column titles.

Grid comments are managed by a set of methods, namely *AddComment*, *DelComment*, *GetComment*, *ShowComment*, *HideComment* and *ClickComment*.

You may lock individual rows and columns using the *LockRows* and *LockColumns* methods. This facility is however not supported in combination with hierarchical rows and/or columns which are specified by *RowTitleDepth* and *ColTitleDepth*.

The Grid can display a *TreeView like* interface on the Row titles. In this mode, the Grid automatically shows and hides row of data as the end user expands and contracts nodes of the tree.

The *RowTreeDepth* property is used to specify the depth of rows in the Grid.

The appearance of the tree is determined by the *RowTreeStyle* property.

User defined bitmaps can be used instead of the default Images by setting the *RowTreeImages* property.

The Grid generates *Expanding* and *Retracting* events when the user interacts with the tree.

The *RowSetVisibleDepth* method can be used to set the visible depth of the tree.

2.24 Group Object

Purpose: This object is used to group a related set of controls together visually, and to impose "radio-button" behaviour.

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm ToolBar
ToolControl

Children Animation Bitmap Button ButtonEdit Calendar Circle ColorButton Combo
ComboEx Cursor DateTimePicker Edit Ellipse Font Grid Group HTMLRenderer Image
ImageList Label List ListView Locator Marker Metafile Poly ProgressBar Rect RichEdit
Scroll SM Spinner Splitter Static SubForm Text Timer TipField TrackBar TreeView
UpDown

Properties Type Caption Posn Size Coord Border Active Visible Event Sizeable Draggable
FontObj FCol BCol Picture CursorObj AutoConf YRange XRange Data Attach EdgeStyle
Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles KeepOnClose Redraw
TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ChooseFont

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles
DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp
MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom
GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap Select

Description

A Group is displayed as an empty box with a border around it whose appearance is defined by the *EdgeStyle* property. The *Caption* property defines a string of text that is displayed in the top left border. The default value is an empty vector.

A Group will be resized if its parent *Form* or Group is resized. It can also be resized directly by the user if its *Sizeable* property is set to 1. By default, when a Group is resized, it automatically adjusts the size and position of its children to maintain the same proportions within it as before. The resizing of a Group and its children can be controlled using the *AutoConf* property or by enabling the *Configure* event (31).

2.25 HTMLRenderer Object

Purpose: The HTMLRenderer Object is a cross-platform mechanism for producing Graphical User Interfaces (GUI), based on HyperText Markup Language (HTML).

Parents ActiveXControl Form Group PropertyPage Root SubForm

Children Timer

Properties Type HTML Posn Size URL Coord Border Active Visible Event Sizeable Moveable SysMenu MaxButton MinButton IconObj Data Attach Translate KeepOnClose AsChild InterceptedURLs CEFVersion Caption AllowContextMenu IsLoading MethodList ChildList EventList PropList

Methods Detach SetZoomLevel GetZoomLevel ExecuteJavaScript WebSocketClose PrintToPDF WebSocketSend ShowDevTools Wait

Events Close Create LoadEnd HTTPRequest WebSocketUpgrade WebSocketReceive WebSocketError DoPopup SelectCertificate

Description

The HTMLRenderer object renders HTML in a window on the screen. It may appear as a top-level window, similar to a Form, or be displayed within another GUI object according to the value of the Boolean *AsChild* property which must be specified when the HTMLRenderer is created. Several HTMLRenderer objects may co-exist in the Dyalog application.

The HTMLRenderer is supported on the following platforms:

- Microsoft Windows
- macOS (ARM-based only)
- Linux (x86_64-based only)

It is not supported on the Raspberry Pi.

The HTMLRenderer is implemented using the [Chromium Embedded Framework \(CEF\)](#). Note that if the **ENABLE_CEF parameter** is set to 0 (its default value is 1) the CEF is disabled and an attempt to create an HTMLRenderer object will fail with an error message.

The HTMLRenderer object can be considered as two components; a client implemented using CEF and an internal server which implements an interface from the APL workspace to the client. The client may communicate with both the internal server and external servers on the web. Thus it can combine and display information from external and internal feeds.

Internal and external communications are distinguished primarily by the *InterceptedURLs* property which specifies which requests from the HTMLRenderer

client component are to be handled by the internal server component and which are to be serviced by the internet. The default value of *InterceptedURLs* has been chosen so that in most cases it can be ignored.

The HTMLRenderer object supports both the HTTP protocol and the WebSocket (WS) protocol.

Using the HTTP protocol, the client requests a resource, such as a style-sheet, an image, or a complete web page, which the server then delivers. All communication is initiated by the client and involves the creating, use, and closing of a TCP/IP socket.

Using WS protocol, the client asks the server for a permanent communications channel (this is done by *upgrading* the TCP/IP socket to a WebSocket) which may subsequently be used for messages initiated by either the client or the server.

The internal server component of the HTMLRenderer is implemented by functions in the workspace.

HTTP protocol communications are handled by callback functions on the *HTTPRequest* event.

WS protocol communications are handled:

- by the *WebSocketUpgrade* event, which reports the initial connection and the WebSocket ID,
- by a callback on the *WebSocketReceive* event
- and by calling the *WebSocketSend* method.

The HTMLRenderer may be initialised by setting its *HTML* property to a character vector representing a base HTML document. This will typically contain references to other documents, such as JavaScript and CSS files which contain code that can influence the way the base HTML is rendered, image files in a variety of formats, and of course hyperlinks to other pages.

If the HTML contains references to other documents, the CEF will retrieve each one by making an HTTP request. Requests with URLs that match a triggering pattern in the *InterceptedURLs* property will generate an *HTTPRequest* event on the HTMLRenderer, which can be directed to a callback function.

Requests with URLs that do not match a pattern in *InterceptedURLs*, or that match a pattern with a 0 in the second column, will be handled by the CEF.

Requests handled by the CEF push the request out to the network to be serviced by an external web server and require that the system has an active internet connection.

An alternative is to initialise the HTMLRenderer by setting its *URL* property. This is typically used to display external content, rather than content delivered from the workspace.

If neither *HTML* nor *URL* is set when the HTMLRenderer is created, it will generate an *HTTPRequest* event with a requested url of `http://dyalog_root`.

When the HTMLRenderer is displayed in its own window, the window caption is set by an assignment to its *Caption* property. The window caption may subsequently change when content is displayed (typically by the title tag in the html). The *Caption* property reports the current window caption.

EXAMPLE

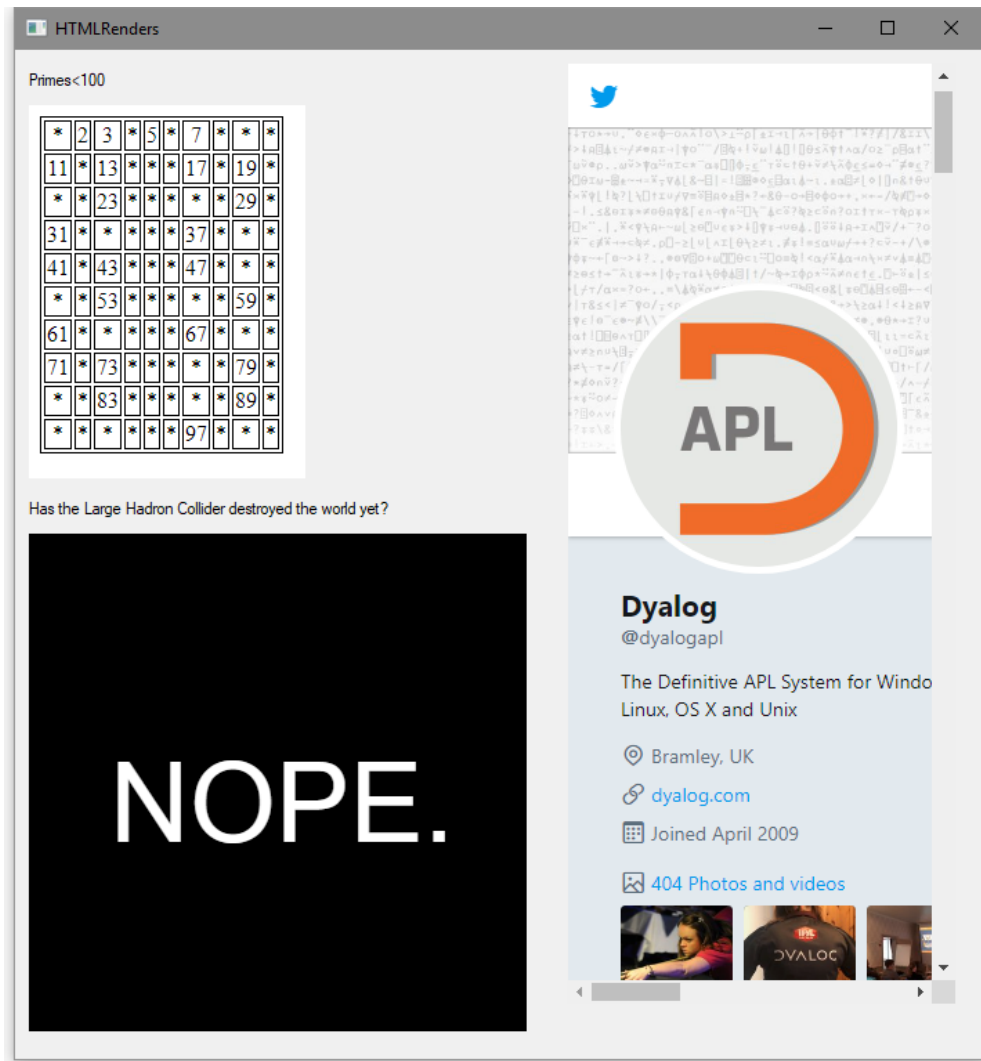
```

▼ Example;Q;U;tw
[1] 'f'⎕WC'Form' 'HTMLRender'
[2] f.(Coord Size)←'Pixel'(730 700)
[3] 'pco'⎕CY'dfns'
[4] 'f.l1'⎕WC'Label' 'Primes<100'(10 10)
[5] 'f.p'⎕WC'HTMLRenderer'('AsChild' 1)('Posn' 40 10)(270 200)
[6] f.p.HTML←HTMLTable('*'@(0∘pco)10 10p1100)
[7] Q←'Has the Large Hadron Collider destroyed the world yet?'
[8] 'f.l2'⎕WC'Label'Q(320 10)
[9] 'f.q'⎕WC'HTMLRenderer'('ASChild' 1)
[10] f.q.(Posn Size)←(350 10)(360 360)
[11] U←'https://hashthelargehadroncolliderdestroyedtheworldyet.com'
[12] f.q.URL←U
[13] tw←'<a class="twtimeline"'
[14] tw,←'href="https://twitter.com/dyalogapl">'
[15] tw,←'Tweets by dyalogapl</a>'
[16] tw,←'<script async src="https://platform.twitter.com/widgets.js"'
[17] tw,←'charset="utf-8"></script>'
[18] 'f.t'⎕WC'HTMLRenderer'('AsChild' 1)
[19] f.t.(Posn Size)←(10 400)(680 280)
[20] f.t.HTML←tw
▼

```

```
enc←{'<',α,'>',(ε⌞ω),'</',((~v\ ' '=α)/α),'>'}
```

```
HTMLTable←{'table border="1"'enc∈(<'tr')enc∈ε"↓(<'td')enc"ω}
```



For more information on the HTMLRenderer, see the [HTMLRenderer User Guide](#).

2.26 Icon Object

Purpose: This object defines an icon.

Parents ActiveXControl CoolBand Form Grid ImageList ListView OLEServer Printer
 ProgressBar PropertyPage PropertySheet RichEdit Root StatusBar SubForm SysTrayItem
 TCPSocket ToolBar ToolControl TrackBar TreeView UpDown

Children Timer

Properties Type File Bits CMap Mask Style KeepBits Size Event Data Handle Accelerator KeepOnClose CBits MethodList ChildList EventList PropList

Methods Detach FileRead FileWrite

Events Close Create Select

Description

The *File* property specifies the name of an icon file (.ICO, .GIF or .PNG), or the name of a DLL or EXE file and the identity of the icon within it.

The *Style* property identifies the size of the icon and must be 'Large' or 'Small'. The former specifies a 32x32 icon and is the default; the latter specifies a 16x16 icon. The size of the icon is not embedded within the icon data, so it is **essential** to specify *Style* correctly. Note that a single file may contain both sizes of an icon. *Style* is only relevant when loading an Icon from file.

If the value of the *File* property is set by *Dyalog APL Language: Ws*, no immediate action is taken, but the corresponding file may subsequently be read or written using the *FileRead* or *FileWrite* methods.

16-bit icons contain fewer than 256 colours and each pixel is either transparent or opaque. The images in such Icons are represented by the *Bits*, *Mask* and *CMap* properties.

32-bit icons are 24-bit images with an 8-bit alpha channel which specifies the degree of transparency of each pixel. The pixels in these Icons are represented by the *CBits* property.

CBits is a rank-2 numeric array whose dimensions represent the rows and columns of pixels in the Icon. The values in *CBits* represent the colour and of each pixel and also its transparency.

Bits is an integer matrix whose elements define the colours of each pixel in the icon in terms of their (0-origin) indices into *CMap*. When the icon is displayed on the screen, the way in which these colours combine with those currently displayed on the screen (the background) is specified by *Mask*. This is a Boolean matrix of the same size as *Bits*. The following table shows how the colour of each resulting pixel is determined.

Bits	Colour	0	Colour
Mask	0	1	1
Pixel	Colour	Background	New Colour

If an element of *Mask* is 0, the corresponding element of *Bits* defines the colour of the resulting pixel that is displayed on the screen. If an element of *Mask* is 1, the resulting pixel that is displayed on the screen is either the current background colour or is a new colour chosen by MS-Windows to be visible against the background. A non rectangular icon is obtained by setting those elements of *Bits* and *Mask* that you want to exclude from the shape to be 0 and 1 respectively.

The size of *Bits* is restricted by the capabilities of the current display driver. *Mask* must have the same shape as *Bits*.

An Icon is **used** by setting the *IconObj* property or *Picture* property of another object to its name or ref.

2.27 Image Object

Purpose: Positions bitmaps and icons within an object.

Parents ActiveXControl Bitmap Form Grid Group Metafile Printer PropertyPage Static StatusBar SubForm ToolBar ToolControl

Children Timer

Properties Type Points Coord Visible Event Draggable Picture OnTop AutoConf Data EdgeStyle Size Accelerator AcceptFiles KeepOnClose MethodList ChildList EventList PropList

Methods Detach

Events Close Create DragDrop MouseDown MouseUp MouseMove MouseDbClick Help Select

Description

The *Points* property specifies the co-ordinates of one or more points at which the specified graphical objects are to be drawn.

The *Picture* property specifies the name(s) of *Bitmap*, *Icon* or *Metafile* object(s) that are to be drawn. It may be a simple character vector or a vector of vectors.

To draw a single graphic picture, the *Picture* property is a simple character vector specifying the name of a *Bitmap*, *Icon* or *Metafile* object. *Points* is either a 2-element vector or a 1-row, 2-column matrix whose elements specify the y-coordinate and x-coordinate respectively at which the object is to be drawn.

To draw the same picture at several different positions, the *Picture* property is a simple character vector specifying the name of the *Bitmap*, *Icon* or *Metafile* object. *Points* is either a 2-column matrix of y-coordinates and x-coordinates, or a nested vector whose first element contains the y-coordinates and whose second element contains the x-coordinates.

To draw several different pictures, the *Picture* property is a vector of character vectors specifying the names of several *Bitmap*, *Icon* and/or *Metafile* objects. *Points* is a 2-column matrix or 2-element nested vector as described above.

Setting the *EdgeStyle* property causes the picture to be surrounded by the appropriate border. For example, setting *EdgeStyle* to 'Plinth' produces a button-like appearance.

Setting the *Size* property causes the picture to be scaled to fit within the specified rectangle. It is only necessary to specify *Size* when an *Image* is used to draw a *Metafile* object. For a *Bitmap* or *Icon*, *Size* defaults to the size of the object being drawn.

The *Dragable* property specifies whether or not the *Image* can be dragged and dropped using the mouse.

EXAMPLES

First make a *Form*

```
'F' □WC 'Form'
```

Then make two *Bitmaps* :

```
'YES' □WC 'Bitmap' 'C:\WDYALOG\WS\YES'  
'NO' □WC 'Bitmap' 'C:\WDYALOG\WS\NO'
```

Display the "YES" *Bitmap* at (20,10)

```
'F.I' □WC 'Image' (20 10)('Picture' 'YES')
```

Display the "YES" *Bitmap* at (20,10) and (20,50)

```
'F.I' □WC 'Image' (20(10 50))('Picture' 'YES')
```

Display the "YES" *Bitmap* at (20,10) and the "NO" *Bitmap* at (20,50)

```
'F.I' □WC 'Image'(20(10 50))('Picture' 'YES' 'NO')
```

2.28 ImageList Object

Purpose: The ImageList object represents a set of bitmapped images.

Parents ActiveXControl ButtonEdit CoolBand CoolBar Form Group ListView OLEServer
PropertyPage Root SubForm TabControl TCPSocket ToolBar ToolControl TreeView

Children Bitmap Cursor Icon Timer

Properties Type Size Event Coord Data Handle Translate ImageCount Masked MapCols
KeepOnClose MethodList ChildList EventList PropList

Methods Detach

Events Close Create

Description

An ImageList object represents an array of bitmapped images which are used to depict items in a *ListView* or *TreeView* object, or the images in a *CoolBar*, *Menu*, *TabControl* or *ToolControl*.

Making an ImageList is a 2-step process. First, you create an (empty) ImageList specifying its *Size* and *Masked* properties. The former establishes the size of each of the bitmapped images in the array. The *Masked* property specifies whether the ImageList is to contain opaque or transparent images. Note that these properties must be established when the ImageList is created by `WC` and may not subsequently be changed using `WS`.

Next, you create a series of *Bitmap* or *Icon* objects as *children* of the ImageList. As you make each one, APL adds the corresponding image (or images) to the ImageList object. If the size of each of the *Bitmap* or *Icon* objects is equal to the *Size* of the ImageList itself, each child object corresponds to an image in the ImageList. However, if you add an object whose *width* is an exact multiple of the *width* of the ImageList, a corresponding number of images will be added.

For example, if the *Size* of the ImageList is 16x16 (the default) and you create a child *Bitmap* of size 16x48, three images (each of size 16x16) will be added to the ImageList. This is more efficient than building the images one-by-one. In other circumstances (where the size of the *Bitmap* or *Icon* is not equal to *Size* of ImageList), the *Bitmap* or *Icon* will be scaled to fit.

Note that when making Bitmaps or Icons as children of an ImageList, it is not necessary to *name* them because they are subsequently referenced only via the *ImageIndex* and

SellImageIndex properties and not by name. The number of images in an *ImageList* is given by the read-only property, *ImageCount*.

The *MapCols* property, which must be specified at the time you create the object, specifies whether or not bitmap colours are remapped to reflect the user's colour preferences.

An *ImageList* is *associated* with a *ListView* or *TreeView* object by the *ImageListObj* property. Each item in the *ListView* or *TreeView* is then allocated a specific image in the *ImageList* by the *ImageIndex* and *SellImageIndex* properties.

2.29 Label Object

Purpose: Displays static text.

Parents ActiveXControl CoolBand Form Grid Group PropertyPage SubForm ToolBar ToolControl

Children Circle Cursor Ellipse Font Marker Poly Rect Text Timer

Properties Type Caption Posn Size Coord Border Justify Active Visible Event Sizeable Draggable FontObj FCol BCol CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj FieldType Decimals FormatString Value Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ChooseFont

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel Select

Description

This object displays a text label, a number, a date or a time value.

If *FieldType* is empty, the Label displays the text defined by its *Caption* property. If the *Caption* property contains one or more linefeed characters (␣) the Label becomes a multi-line Label which is top-left justified and automatically wraps on white-space characters (such as space and tab) to fit in the width provided.

If *FieldType* is 'Numeric', 'LongNumeric', 'Currency', 'Date', 'LongDate', or 'Time' the Label converts and formats the number defined by its *Value* property and displays this instead. See *FieldType* property for details.

The *Border* property determines whether or not the label has a border. A value of 0 means no border (the default). A value of 1 means that a 1-pixel border is drawn around the label.

By default, the value of the *EdgeStyle* property for a Label is 'None' and the value of *BCol* is 0 which is normally white, or grey if the parent object has a 3-dimensional appearance. You can change its appearance by setting *EdgeStyle* and/or *BCol* to different values.

2.30 List Object

Purpose: Allows the user to select one or more items from a list.

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm ToolBar ToolControl

Children Circle Cursor Ellipse Font Marker Poly Rect Text Timer

Properties Type Items Posn Size Style Coord Border Active Visible Event VScroll SellItems Sizeable Draggable FontObj FCol BCol CursorObj AutoConf Index Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles MultiColumn ColumnWidth KeepOnClose Redraw SortItems TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ChooseFont

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap Select

Description

The *Items* property is either a vector of character vectors or a character matrix, and determines the items in the List.

The size and position of the area used to display the list is defined by *Size* and *Posn*. If *Size* is not chosen to represent an exact number of lines of text, the bottom line of text may be clipped.

The *Index* property specifies or reports the position of *Items* in the list box as a positive integer value. If *Index* has the value "n", it means that the "nth" item in *Items* is displayed on the top line in the list box. However, it is ignored if all the *Items* fit within

the List object. Note that *Index* can only be set using *Dyalog APL Language: Ws* and not by *Dyalog APL Language: Wc*. The default value for *Index* is `⌈IO`.

The *Style* property may be 'Single' (the default) or 'Multi'. 'Single' allows only a single item to be selected. 'Multi' allows several items to be chosen. In either case, if the *Select* event is enabled, it is generated whenever the selection changes. If *Style* is 'Multi' the List will generate a *Select* event every time an item is added to the selected list.

Under Windows, you may select or de-select multiple items in a List object by pressing the Ctrl key at the same time as pressing the left mouse button.

The *SelItems* property is a Boolean vector with one element per element or row in *Items* and indicates which (if any) of the items is currently selected (and highlighted).

The *VScroll* property determines whether or not the list has a scrollbar. Its possible values are :

<code>^-2</code>	scrollbar if required
<code>^-1</code>	scrollbar if required
<code>0</code>	no scrollbar

Note that data in a List is always scrollable if there are more items than will fit in the box. *VScroll* determines ONLY whether or not a scrollbar is provided.

The *MultiColumn* property is a Boolean value that specifies whether or not the List object displays its items in columns. The default is 0 which produces a single-column display. If *MultiColumn* is 1, the List object displays its items in columns whose width is defined by the *ColumnWidth* property.

2.31 ListView Object

Purpose: The ListView object displays a collection of items.

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm ToolBar
ToolControl

Children Bitmap Circle Cursor Ellipse Font Icon ImageList Marker Poly Rect Text Timer

Properties Type Items Posn Size Style Coord Align Border Active Visible Event
DragItems View AutoArrange Header Wrap EditLabels ImageListObj ReportInfo ColTitles
ImageIndex ReportImageIndex ReportBCol HeaderImageList HeaderImageIndex VScroll

HScroll SellItems Sizeable Dragable FontObj FCol BCol CursorObj AutoConf Data Attach
 EdgeStyle Handle Hint HintObj Tip TipObj ColTitleAlign ColTitle3D Translate Accelerator
 AcceptFiles KeepOnClose CheckBoxes FullRowSelect GridLines Redraw TabIndex
 AlwaysShowSelection ItemGroups ItemGroupMetrics MethodList ChildList EventList
 PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj
 GetItemState SetItemState GetItemPosition

Events Close Create FontOK FontCancel BeginEditLabel EndEditLabel DragDrop
 Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus
 LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter
 MouseLeave MouseWheel GestureZoom GesturePan GestureRotate
 GestureTwoFingerTap GesturePressAndTap ColumnClick SetItemPosition ItemDown
 ItemUp ItemDbClick GetTipText SetColSize Select

Description

The `ListView` object is a window that displays a collection of items, each item consisting of an icon and a label. The `ListView` provides several ways of arranging items and displaying individual items. For example, additional information about each item can be displayed in columns to the right of the icon and label. An example of the use of a `ListView` object is the "My Computer" Windows utility.

The `Items` property is a vector of character vectors that specifies the labels for the items displayed by the `ListView`. The `ImageListObj` property specifies the names of two `ImageList` objects that define two sets of icons; a large icon (32x32 pixel) set and a small icon (16x16 pixel) set. Alternatively, `ImageListObj` may be empty (no icons displayed) or contain just the name of a single large icon `ImageList`.

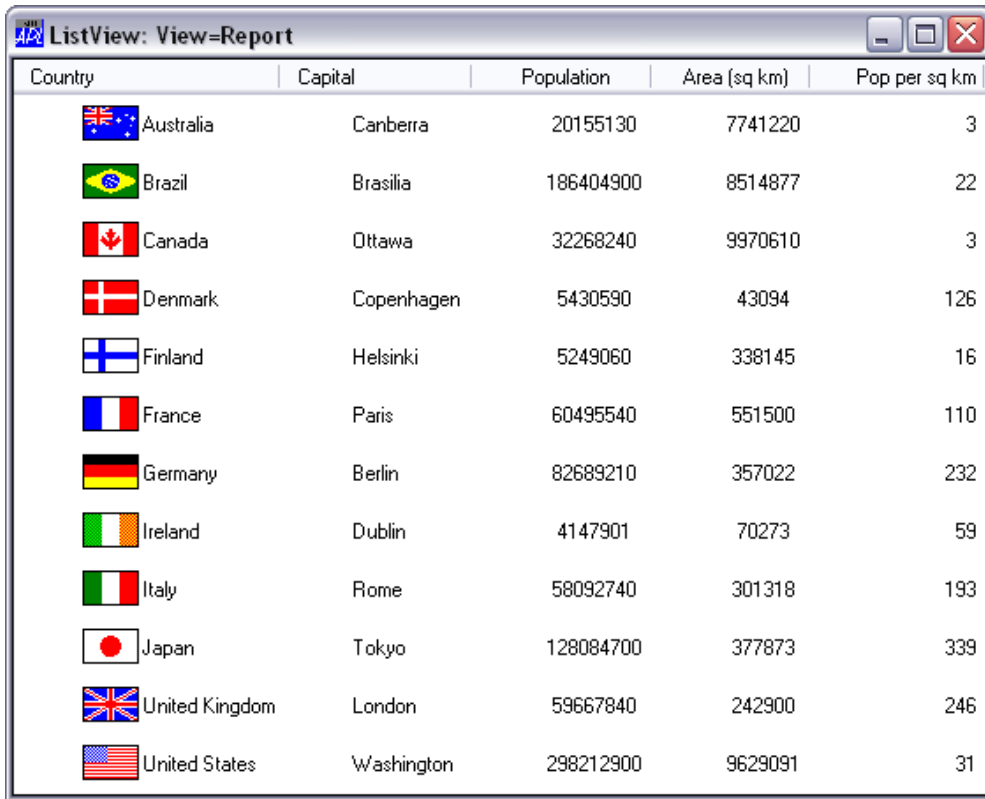
The `View` property contains a character vector that determines how the items are displayed. It may have one of the following values; 'Icon' (the default), 'SmallIcon', 'List' or 'Report'. When `View` is 'Icon' or 'SmallIcon', the items are arranged row-wise with large or small icons as appropriate. When `View` is set to 'List', the items are arranged *column-wise* using small icons. Examples of 'Icon' and 'List' views are illustrated below.



When View is set to 'Report', the items are displayed in a single column using small icons but with the matrix specified by *ReportInfo* displayed alongside. In this format, the Boolean *Header* property determines whether or not the object also provides column headings. Its default value is 1. The column headings themselves are specified by the *ColTitles* property. Their alignment (and the alignment of the data in the columns beneath them) is defined by the *ColTitleAlign* property.

The appearance of the column titles is further controlled by the *ColTitle3D* property. This is a Boolean value (default 1) which specifies whether or not the column titles have a 3-dimensional (plinth) appearance. *Header* and *ColTitle3D* may only be set when the object is created using `WC` and may not subsequently be changed by `WS`.

In 'Report' View, columns may be resized by the user dragging the bars between the titles, or under program control using the `SetColSize` event. A 'Report' view example is illustrated below.



Country	Capital	Population	Area (sq km)	Pop per sq km
 Australia	Canberra	20155130	7741220	3
 Brazil	Brasilia	186404900	8514877	22
 Canada	Ottawa	32268240	9970610	3
 Denmark	Copenhagen	5430590	43094	126
 Finland	Helsinki	5249060	338145	16
 France	Paris	60495540	551500	110
 Germany	Berlin	82689210	357022	232
 Ireland	Dublin	4147901	70273	59
 Italy	Rome	58092740	301318	193
 Japan	Tokyo	128084700	377873	339
 United Kingdom	London	59667840	242900	246
 United States	Washington	298212900	9629091	31

The *DragItems* property is Boolean and specifies whether or not the user may drag an item from one position to another. Its default value is 1 (dragging is enabled).

The *AutoArrange* property is Boolean and specifies whether or not the items are automatically re-arranged whenever an item is repositioned by the user or moved under program control. Its default value is 0.

The *EditLabels* is a Boolean property (default 0) that determines whether or not the user may edit the labels which are specified by the *Items* property.

The *Style* property may be 'Single', which specifies that only one item may be selected at a time, or 'Multi' which permits multiple selections to be made. The default is 'Multi'.

The *CheckBoxes* property is Boolean and specifies whether or not check boxes are drawn to the left of items. Its default value is 0.

The *GridLines* property is Boolean and specifies whether or not grid lines are drawn between items. This applies only when *View* is 'Report'. Its default value is 0.

In 'Report' View, the *ReportImageIndex* property specifies images to be displayed alongside each item, and overrides the images specified by *ImageListObj*. The *HeaderImageList* and *HeaderImageIndex* properties specify images to be displayed alongside each column title.

The following example illustrates the use of these properties.

Country	Capital	08:00	14:00	20:00	02:00
Australia	Canberra	Few Clouds	Very Cloudy	Thunder Storm	Scattere...
Brazil	Brasilia	Cloudy	Very Cloudy	Clear	Light Rain
Canada	Ottawa	Scattered Cloud	Few Clouds	Light Rain	Cloudy
Denmark	Copenhagen	Cloudy	Very Cloudy	Cloudy	Scattere...
Finland	Helsinki	Scattered Cloud	Clear	Scattered Cloud	Scattere...
France	Paris	Rain	Few Clouds	Very Cloudy	Light Rain
Germany	Berlin	Clear	Clear	Clear	Light Rain
Ireland	Dublin	Cloudy	Thunder Storm	Very Cloudy	Light Rain
Italy	Rome	Thunder Storm	Light Rain	Few Clouds	Light Rain
Japan	Tokyo	Thunder Storm	Scattered Cloud	Light Rain	Thunder ...
United Kingdom	London	Clear	Scattered Cloud	Thunder Storm	Rain
United States	Washington	Clear	Few Clouds	Cloudy	Rain

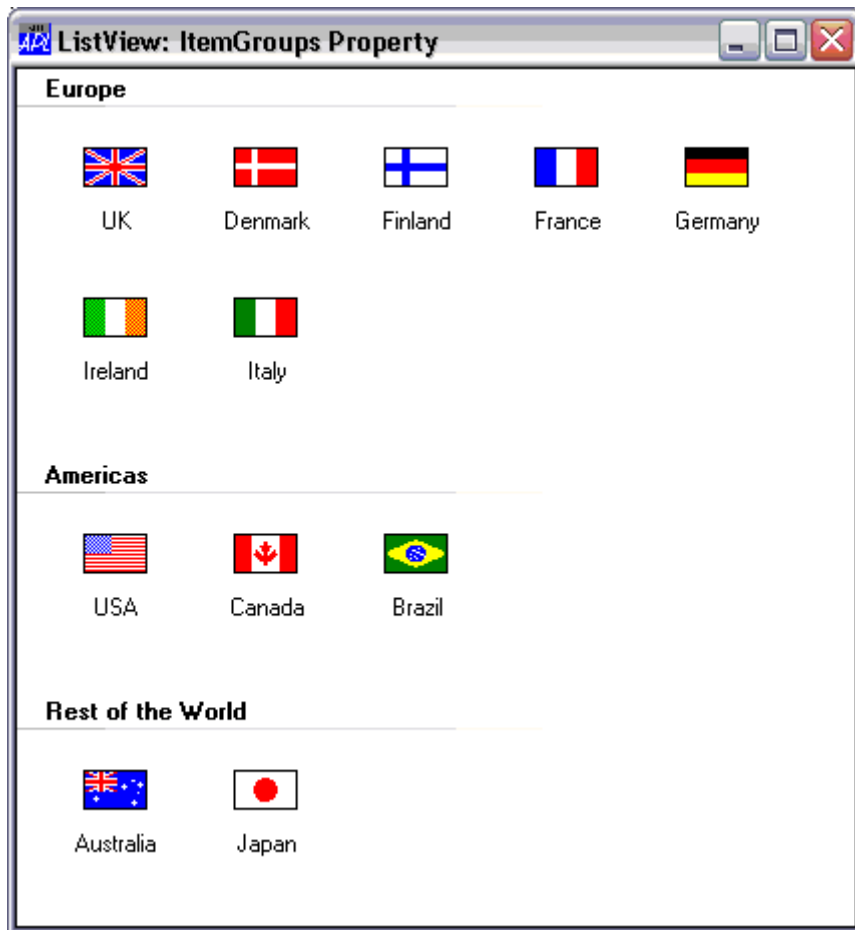
The *ReportBCol* property specifies each item's background colour.

The *FullRowSelect* property is Boolean and specifies whether or not the entire row is highlighted to indicate selected items. This applies only when View is 'Report'. Its default value is 0.

The *ItemGroups* and *ItemGroupMetrics* properties allow you to display items in groups as illustrated below.

Note

This feature only applies if *Native Look and Feel* is enabled.



2.32 Locator Object

Purpose: Allows the user to input a point, ellipse, line or rectangle.

Parents ActiveXControl Form Group PropertyPage PropertySheet Root Static SubForm
TCPSocket ToolBar ToolControl

Children Timer

Properties Type Posn Size LStyle Style Coord Event Step Sizeable CursorObj Data
Accelerator KeepOnClose MethodList ChildList EventList PropList

Methods Detach Wait

Events Close Create Locator Select

Description

This object is used to obtain graphical input from the user. Like a pop-up menu or a *MsgBox*, the Locator is a *modal* object whose interaction with the user is initiated by a "local" *Dyalog APL Language: Dq*.

This is terminated when the user releases a mouse button or presses any key other than a cursor movement key, Shift, Ctrl or Alt. It is usual to initiate the *Dyalog APL Language: Dq* for the Locator from within a callback function attached to a *MouseDown (1) Event*.

When the "local" *Dyalog APL Language: Dq* is terminated, a *Locator (80) Event* is generated. The associated event message contains the new position and size of the Locator, together with how the event was generated (keystroke or mouse button). To obtain the Locator's new position or size, you **must** enable the event by setting its "action" code to 1, or to the name of a suitable callback function.

The value of the *Style* property determines the type of locator displayed. It may be 'Point', 'Line', 'Rect', or 'Ellipse'. The default value is 'Rect'. The value of the *Sizeable* property is 0 or 1 and determines whether or not "rubberbanding" is enabled. Its default value is 1 which turns "rubberbanding" on. The *Size* property determines the initial size of the Locator when displayed by *Dyalog APL Language: Dq*. Its default value is (0,0).

If *Style* is 'Rect' the Locator displays a rectangle. One corner of the rectangle is positioned at *Posn*. The diagonally opposite corner is positioned at (*Posn*+*Size*). If *Sizeable* is 0, the entire rectangle is dragged as the mouse is moved. If *Sizeable* is 1, the corner initially defined by (*Posn*+*Size*) is dragged (rubberbanding the rectangle) as the mouse is moved. The rectangle disappears when the operation is terminated. The new position or size of the rectangle is reported in the *Locator* event message.

If *Style* is 'Ellipse' the Locator displays an ellipse. One corner of the bounding rectangle of the ellipse is positioned at *Posn*. The diagonally opposite corner is positioned at (*Posn*+*Size*). If *Sizeable* is 0, the entire ellipse is dragged as the mouse is moved. If *Sizeable* is 1, the corner of the bounding rectangle initially defined by (*Posn*+*Size*) is dragged (rubberbanding the ellipse) as the mouse is moved. The ellipse disappears when the operation is terminated. The new position or size of the bounding rectangle of the ellipse is reported in the *Locator* event message.

If *Style* is 'Line' the Locator displays a line drawn between the points defined by *Posn* and *Posn*+*Size*. If *Sizeable* is 0, the line is dragged with the cursor as the mouse is moved. If *Sizeable* is 1, the end of the line initially defined by *Posn*+*Size* is dragged (rubberbanding the line) as the mouse is moved. The line disappears when the

operation is terminated. The new position or size of the line is reported in the *Locator* event message.

If 'Style' is 'Point', the values of *Sizeable* and *Size* are ignored. During the *Dyalog APL Language: Dq* no visible feedback (other than the cursor) is provided as the user moves the mouse. When the *Dyalog APL Language: Dq* terminates, the new position of the Locator is reported in the *Locator* event message.

The *Step* property is a 2-element integer vector (default value 1 1) that specifies the increments (in pixels) by which the size or position of the Locator changes in the Y and X directions respectively as the user moves the Locator.

The Locator is normally initiated from a *MouseDown* (1) event, and it is natural to place it at the current cursor position. However, if you are using rubberbanding, you will normally want to have the cursor appear at the end or corner of the Locator that moves. If you start with a non-zero sized Locator, you must set *Posn* (which defines the **fixed** end or corner) to the current cursor position minus *Size* to achieve this effect.

2.33 Marker Object

Purpose: A graphical object used to draw polymarkers.

Parents ActiveXControl Animation Bitmap Button ButtonEdit Combo ComboEx Edit Form Grid Group Label List ListView MDIClient Metafile Printer ProgressBar PropertyPage PropertySheet RichEdit Scroll Spinner Static StatusBar SubForm TabBar TipField ToolBar TrackBar TreeView UpDown

Children Timer

Properties Type Points Style Size FCol Coord Visible Event Dragable OnTop AutoConf Data Accelerator KeepOnClose DrawMode MethodList ChildList EventList PropList

Methods Detach

Events Close Create DragDrop MouseDown MouseUp MouseMove MouseDbIClick Help Select

Description

The *Points* property specifies one or more sets of points at which one or more sets of polymarkers are to be drawn.

The *Style* property determines the symbol that is drawn at each of a set of points. Marker styles are specified either by numbers which represent the following symbol shapes.

0	.
1	+
2	*
3	□
4	×
5	◇
6	◦

or by character vectors containing the names of *Bitmap* or *Icon* objects.

The height of each symbol is specified by the value of the *Size* property. However this applies only to *Styles* 1-6 and is ignored if *Style* is 0 or the name of a *Bitmap*. The colour of each symbol is specified by the *FCol* property. The default is black.

The value of *Dragable* determines whether or not the object can be dragged. The value of *AutoConf* determines whether or not the Marker object is resized when its parent is resized.

The structure of the property values is best considered separately for single and multiple sets of polymarkers.

Single Set of Polymarkers

For a single set of polymarkers, *Points* is either a 2-column matrix of (y,x) co-ordinates, or a 2-element vector of y and x co-ordinates respectively.

Style and *Size* are both simple scalar numbers.

FCol is either a single number representing a standard colour, or a 3-element vector which specifies the marker colour explicitly in terms of RGB values.

EXAMPLES

First make a *Form*:

```
'F' □WC 'Form'
```

Draw a point at (y=20, x=10):

```
'F.M1' □WC 'Marker' (20 10)
```

Draw a row of points at (y=20, x=10, 20, ... 90) : (Note scalar extension of y-coordinate)

```
'F.M1' □WC 'Marker' (20(10×19))
```

Draw "+" symbols at each corner of a box:

```
Y ← 10 10 50 50
X ← 10 50 50 10
'F.M1' □WC 'Marker' (Y X) 1
```

Ditto, but draw them 10% high:

```
'F.M1' □WC 'Marker' (Y X) 1 10
```

Ditto, but use "*" symbols in green:

```
'F.M1' □WC 'Marker' (Y X) 2 10 (0 255 0)
```

Multiple Sets of Polymarkers

To draw multiple sets of polymarkers with a single name, *Points* is a nested vector whose items are themselves 2-column matrices or 2-element nested vectors.

Style and *Size* may be simple scalars specifying a single type and/or size of symbol to be used for all the sets of polymarkers, or vectors specifying different symbols and/or sizes for each set.

FCol may be a single number or a single (enclosed) 3-element vector applying to all the sets of polymarkers. Alternatively, *FCol* may be a vector whose elements refer to each of the sets of polymarkers in turn. If so, the elements may be single numbers or nested RGB triplets, or a combination of the two.

EXAMPLES

First make a *Form*:

```
'F' □WC 'Form'
```

Draw a "□" at (10,20) and a "◇" at (20,20):

```
'F.M1' □WC 'Marker' ((1 2p10 20)(1 2p20 20)) (3 5)
```

Draw "+" symbols at each corner of one box and "o" symbols at each corner of another:

```
Y1 X1 ← (10 10 50 50) (10 50 50 10)
Y2 X2 ← (20 20 40 40) (20 40 40 20)
'F.M1' WC 'Marker' ((Y1 X1)(Y2 X2)) (1 6)
```

Ditto, but draw the "+" symbols with height 2% and the "o" symbols 5%:

```
'F.M1' WC 'Marker' ((Y1 X1)(Y2 X2)) (1 6) (2 5)
```

Ditto, but draw the "+" symbols in red and the "o" symbols in blue:

```
'F.M1' WC 'Marker' ((Y1 X1)(Y2 X2)) (1 6) (2 5)
('FCol' (255 0 0)(0 0 255))
```

2.34 MDIClient Object

Purpose: Implements Multiple Document Interface (MDI) behaviour.

Parents ActiveXControl Form SubForm

Children Circle Ellipse Font Marker Poly Rect SubForm Text Timer

Properties Type Posn Size Coord Border Event BCol Picture IconObj CursorObj YRange XRange Data Attach EdgeStyle Handle MDIActive MDIActiveObject Hint HintObj Tip TipObj Translate Accelerator KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach MDICascade MDITile MDIArrange GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap Select

Description

The multiple-document interface (MDI) is a document-oriented interface that is commonly used by word-processors, spreadsheets and other applications that deal with *documents*. An MDI application allows the user to display multiple documents at the same time, with each document displayed in its own window.

The MDIClient object is a container object that effectively specifies the client area within the parent *Form* in which the *SubForm* are displayed. The MDIClient object also imposes special MDI behaviour which is quite different from that where a *SubForm* is simply the child of another *Form*.

By default, the MDIClient occupies the entire client area within its parent *Form*. This is the area within the *Form* that is not occupied by *CoolBars*, *MenuBar*, *ToolBar*, *ToolControls*, *TabBars*, *TabControls* and *StatusBars*. In most applications it is therefore not necessary to specify the position and size of the MDIClient object, although you may do so if you want to reserve additional space in the parent *Form* for other objects.

Each of the four sides of an MDIClient object is automatically *attached* to the corresponding side of its parent *Form* and maintains its position when the parent *Form* is resized. This means that a default MDIClient always occupies the entire client area of its parent *Form*, regardless of how the parent is resized.

The appearance of the MDIClient may be changed using its *Border*, *BColor* and *Picture* properties. The *EdgeStyle* property has no direct effect and is provided only to pass on a value to its child *Forms*.

The *MDIActive* and *MDIActiveObject* properties contain the name of and a ref to the *SubForm* that currently has the focus. You may set these properties as well as query them.

You can call methods which cause the MDIClient to organise its child *SubForms* in some way. These methods are as follows:

<i>MDICascade</i>	Causes the MDIClient to organise its child <i>Forms</i> in an overlapping manner.
<i>MDITile</i>	Causes the MDIClient to arrange its child <i>Forms</i> as a row or column.
<i>MDIArrange</i>	Causes the MDIClient to arrange the icons associated with any minimised child <i>Forms</i> in an orderly fashion.

2.35 Menu Object

Purpose: This is a pop-up object which allows the user to initiate an action or to select an option using a "menu".

Parents ActiveXControl Calendar CoolBand CoolBar DateTimePicker Form Grid Menu
MenuBar OLEServer Root StatusField SubForm SysTrayItem TCPSocket ToolBar
ToolControl

Children Bitmap Menu MenuItem Separator Timer

Properties Type Caption Posn Coord Align Active Event FontObj FCol BCol BtnPix Data
EdgeStyle Handle Translate Accelerator KeepOnClose ImageListObj ImageIndex
MethodList ChildList EventList PropList

Methods Detach Wait

Events Close Create DropDown Select

Description

For a Menu that is owned by a *MenuBar* or another Menu, the *Caption* property determines the text string that is displayed as the "choice". The Menu is then popped up by the user clicking on this text. It is automatically popped down when the user chooses an option (by selecting a *MenuItem*) or cancels the operation (by clicking elsewhere).

If a Menu belongs to a *Form*, *SubForm* or is a top-level object, it must be popped up by the application. This is commonly done in response to a *MouseDown* event. A Menu is popped-up by calling *Dyalog APL Language: Dq* with only the name of the Menu as its argument. The user may therefore not interact with any other object until a selection is made or until the operation is cancelled. When either occurs, the Menu is automatically popped down and de-activated, and its *Dyalog APL Language: Dq* terminates.

The Menu object does not have a *Size* property. Instead, its size is determined automatically by its contents.

If a Menu is owned by a *MenuBar* or by another Menu, its position within its parent is also calculated automatically, dependent on the order in which other related objects are established. The *Posn* property may however be used to **insert** a new Menu into an existing structure. For example, having defined three Menu objects as children of a *MenuBar*, you can insert a fourth one between the first and the second by specifying its *Posn* to be 2. Note that the value of *Posn* for the Menus that were previously second and third will then be reset to 3 and 4 respectively.

If a Menu is a child of a *MenuBar* which is itself a child of a *Form* or *SubForm*, the *Align* property can be set to 'Right'. This is used to position a single Menu (or *MenuItem*) at

the rightmost end of a *MenuBar*. This does not apply if the *MenuBar* is owned by a *ToolControl*.

The *BtnPix* property is used to display a picture in a Menu. *BtnPix* specifies the names of, or refs to, three *Bitmap* objects. The first *Bitmap* is displayed when the Menu does not have the focus (normal), the second when it does have the focus (highlighted). The third *Bitmap* is displayed when the Menu is made inactive (*Active* property is 0). If *Caption* is also defined, it is displayed **on top of** the bitmaps.

If the Menu is a submenu (owned by a Menu), you may set its *EdgeStyle* property to 'Plinth'. This causes the Menu to take on an appearance that is similar to a pushbutton and be raised when not selected and recessed when selected. Note that to enable 3-dimensional appearance, you must set *EdgeStyle* to something other than 'None' for all the objects above the Menu in the tree.

EdgeStyle, *BtnPix*, *Font*, *FCol* and *BCol* do not affect the appearance of a Menu if it is the direct child of a *MenuBar*. However, the *EdgeStyle* property **must** be set to something other than 'None' if you want its children Menu and *MenuItem* objects to have a 3-dimensional appearance.

2.36 MenuBar Object

Purpose: Specifies a horizontal menu bar displayed at the top of a *Form* .

Parents ActiveXControl Form SubForm ToolControl

Children Bitmap Menu MenuItem Separator Timer

Properties Type Visible Event FontObj Data EdgeStyle MDIMenu Handle Translate KeepOnClose MethodList ChildList EventList PropList

Methods Detach

Events Close Create

Description

Unless it is made invisible the *MenuBar* is always available to the user to initiate actions or to select options. A *MenuBar* has a fixed position and size.

It is possible to have more than one *MenuBar* associated with the same *Form* or *SubForm*, but only one of them should be *Visible* at any one time.

The following example illustrates how a menu structure can be built up from a `MenuBar`. For clarity, the example is indented, and the definition of the `Event` property is omitted.

```
'F'           □WC 'Form' 'Menu Example'
'F.M'         □WC 'MenuBar'
'F.M.FILE'    □WC 'Menu' '&File'
'F.M.MAT'     □WC 'Menu' '&Materials'
'F.M.MAT.B'   □WC 'MenuItem' '&Brick'
'F.M.MAT.C'   □WC 'MenuItem' '&Concrete'
'F.M.MAT.S'   □WC 'MenuItem' '&Stone'
'F.M.MAT.SEP' □WC 'Separator'
'F.M.MAT.W'   □WC 'Menu' '&Wood'
'F.M.MAT.W.O' □WC 'MenuItem' '&Oak'
'F.M.MAT.W.T' □WC 'MenuItem' '&Teak'
'F.M.MAT.W.M' □WC 'MenuItem' '&Mahogany'
```

Note that putting a *Separator* (either *Style*) in a `MenuBar` has the effect of breaking the bar vertically, that is, the next *Menu* or *MenuItem* you add will appear on the left-hand side on the line below.

The `EdgeStyle` property has no effect on the appearance of a `MenuBar` or of a direct child of a `MenuBar`. However, if you want the sub-menus to have a 3-dimensional appearance, you **must** set the `EdgeStyle` property of the `MenuBar` to something other than `'None'`.

If the `MenuBar` is owned by a *Form* that is the parent of an *MDIClient*, you can set the `MDIMenu` property to the name of the *Menu* you wish to nominate as the *window* menu. This menu will automatically be updated with the *Captions* of the child *SubForm* and may be used to select the currently active one.

2.37 MenuItem Object

Purpose: This object allows the user to initiate an action or to select an option from a menu.

Parents Menu MenuBar

Children Bitmap Timer

Properties Type Caption Posn Style Align Active Event Checked FontObj FCol BCol
BtnPix Data EdgeStyle Hint HintObj Tip TipObj Translate Accelerator KeepOnClose
ImageIndex MethodList ChildList EventList PropList

Methods Detach**Events** Close Create Select**Description**

The *Caption* property determines the text string that is displayed in its parent as the menu option. The size of a MenuItem is determined by the size of its *Caption*, or by the size of the largest object (*Menu*, MenuItem or *Separator*) with the same parent. The position of the MenuItem is normally determined by the order in which it is created in relation to other objects with the same parent. However, you can use the *Posn* property to **insert** a new MenuItem into an existing structure. For example, having defined three MenuItem objects as children of a *Menu*, you can insert a fourth one between the first and the second by specifying its *Posn* to be 2. Note that the value of *Posn* for the MenuItems that were previously second and third will then be reset to 3 and 4 respectively.

The *Style* property may be 'Check' (the default) or 'Radio'. *Style* determines the type of graphic displayed alongside the *Caption* if the MenuItem is *checked*.

The *Checked* property is a single number with the value 0 or 1. 0 means not checked (the default), 1 means checked and a tick mark or radio dot is placed alongside its *Caption*. This property is frequently used to indicate which of a choice of options is currently set.

If a MenuItem is a child of a *MenuBar* which is itself a child of a *Form* or *SubForm*, the *Align* property can be set to 'Right'. This is used to position a single MenuItem (or *Menu*) at the rightmost end of a *MenuBar*. This does not apply if the *MenuBar* is owned by a *ToolControl*.

If you set the *EdgeStyle* property to 'Plinth', the MenuItem will take on an appearance that is similar to a pushbutton and be raised when not selected and recessed when selected. Note that to enable 3-dimensional appearance, you must set *EdgeStyle* to something other than 'None' for all the objects above the MenuItem in the tree.

The *BtnPix* property is used to display a picture in a MenuItem. *BtnPix* specifies the names of, or refs to, three *Bitmap* objects. The first *Bitmap* is displayed when the MenuItem does not have the focus (normal), the second when it does have the focus (highlighted). The third *Bitmap* is displayed when the MenuItem is made inactive (*Active* property is 0). If *Caption* is also defined, it is displayed **on top of** the bitmaps.

Alternatively, you may display an image alongside the *Caption* using the *ImageIndex* property. This selects a picture from the *ImageList* associated with the *ImageListObj* property of the parent *Menu*.

EdgeStyle, *BtnPix*, *FontObj*, *FCol* and *BCol* are not effective if the *MenuItem* is the direct child of a *MenuBar*.

A *MenuItem* generates a *Select* event (if enabled) when the user chooses it.

2.38 Metafile Object

Purpose: This object represents a picture in Windows Metafile format.

Parents ActiveXControl Bitmap CoolBand Form Group OLEServer Printer PropertyPage PropertySheet Root Static SubForm TCPSocket ToolBar ToolControl

Children Circle Ellipse Font Image Marker Poly Rect Text Timer

Properties Type File Size Coord RealSize Event YRange XRange Data Handle Translate Accelerator KeepOnClose MethodList ChildList EventList PropList

Methods Detach FileRead FileWrite

Events Close Create Select

Description

The Windows Metafile is a mechanism for representing a picture in terms of a collection of graphical components. Windows Metafiles are distributed in special files (.WMF) from which they are loaded into memory for use by an application. Once loaded a Metafile is a Windows "resource" that can be used in a variety of ways. The Metafile object represents this resource.

The *File* property specifies the name of a .WMF file from which the Metafile is to be loaded or to which it is to be saved. If you specify *File* with *Dyalog APL Language: Wc* the Metafile object is loaded from it. If you specify *File* with *Dyalog APL Language: Ws* no action takes place until you instruct the Metafile object to re-initialise itself from the file or to save itself to the file. These operations are performed using the *FileRead* and *FileWrite* methods. If you omit the *File* property in the argument to *Dyalog APL Language: Wc* or if you specify a null vector, the Metafile object is initially empty. The following example loads the picture defined by the GOLF.WMF Metafile that is distributed with Microsoft Office.

```
'GOLF' ⎕WC 'Metafile' 'C:\MSOFFICE\CLIPART\GOLF'
```

Whether or not the Metafile object is initialised from a file, you can add graphical components to it by creating child objects. However the Metafile behaves like a *Bitmap* object in that its children cannot be modified using *Dyalog APL Language: Ws* nor can they be removed using `ⓘEX`. The components of a Metafile that has been initialised from a .WMF file also cannot be referenced in any way. It is therefore recommended that you use unnamed objects when you create the graphical components of a Metafile.

The following statements create an empty Metafile called MF and then draw a line and circle in it.

```
'MF' ⓂWC 'Metafile'
'MF.' ⓂWC 'Poly' (50(10 90))
'MF.' ⓂWC 'Circle' (50 50) 30
```

Like the *Bitmap*, *Icon*, *Font* and *Cursor* objects, the Metafile is a resource that is not visible until it is *used*. This is done by setting the *Picture* property of another object (*Form*, *Image*, *Static* or *SubForm*) to the name of, or ref to, the Metafile object. For example, to display the Metafile MF in a *Form*, you could type :

```
'TEST' ⓂWC 'FORM' ('Picture' 'MF')
```

You can also copy a Metafile object to the Windows Clipboard from where it can be pasted into another application. This is done by creating a *Clipboard* object and then setting its *MetafileObj* property to the name of the Metafile object to be exported. For example :

```
'CL' ⓂWC 'Clipboard'
'CL' ⓂWS 'MetafileObj' 'MF'
```

The *FileWrite* method may be used to save a Metafile object on a file. The following statements save the Metafile MF in a file called TEST.WMF.

```
MF.File←'TEST'
MF.FileWrite
```

The *Size* property determines the granularity of the Metafile. Its default value is the size of its parent. If you intend to replay the Metafile at higher resolution, you should set *Size* accordingly.

The *RealSize* property specifies the suggested size of a Metafile in units of 0.01mm. Setting *RealSize* has the effect of making the Metafile *placeable*. Certain programs (such as Word for Windows) only support placeable metafiles.

2.39 MsgBox Object

Purpose: Provides a "modal" dialog box for displaying messages, errors, warnings and other information. The dialog box has a title, one or more lines of text, and up to three buttons.

Parents ActiveXControl Calendar CoolBand DateTimePicker Form Grid OLEServer PropertyPage PropertySheet Root SubForm TCPSocket ToolBar ToolControl

Children Timer

Properties Type Caption Text Style Btns Default Event Data EdgeStyle KeepOnClose MethodList ChildList EventList PropList

Methods Detach Wait

Events Close Create MsgBtn1 MsgBtn2 MsgBtn3

Description

The *Caption* property determines the text displayed in the object's title bar.

The *Text* property determines the text to be displayed as the message.

The *Style* property determines the type of icon which is displayed. This is a character vector with one of the following values:

'Msg '	no icon (the default)
'Info '	information message icon
'Query '	query (question) icon
'Warn '	warning icon
'Error '	critical error icon

The *Btns* property determines the set of buttons to be displayed. It is a simple vector (one button) or a matrix with up to 3 rows, or a vector of up to 3 character vectors specifying the captions for up to 3 buttons. MS-Windows restricts you to a fixed set of button captions which are described below. However, the property has been designed more generally to be useful under different GUIs and perhaps later revisions of Windows. The buttons are arranged along the bottom of the dialog box in the order specified.

The *Btns* property may specify one of six sets of buttons as follows:

- 'OK '
- 'OK ' 'CANCEL '
- 'RETRY' 'CANCEL '
- 'YES' 'NO '
- 'YES' 'NO' 'CANCEL '
- 'ABORT' 'RETRY' 'IGNORE '

If any other combination is specified, *Dyalog APL Language: Wc* and *Dyalog APL Language: Ws* will report a DOMAIN ERROR. The names of the buttons are however case-insensitive, so the system will accept 'ok ', 'Ok ', 'oK ' or 'OK '. If *Btns* is not specified, it assumes a default according to *Style* as follows:

Style	Btns
'Msg' or 'Info'	'OK '
'Warn' or 'Error'	'OK ' 'CANCEL '
'Query'	'YES' 'NO '

The *Default* property may be used to determine which of the buttons is the "default" button, that is, the one which initially has the focus and is "selected" when the user presses the Enter key. It has the value 1, 2 or 3. If *Default* is not specified, the first button is the "default" button. Note that if the user switches focus to another button and presses Enter, this action selects the button with the focus.

Like a pop-up (floating) *Menu*, the *MsgBox* object is unusual in that it is strictly modal. It is created by *Dyalog APL Language: Wc* in the normal way, but at that stage is invisible and inactive. It is activated ONLY when *Dyalog APL Language: Dq* is called with the name of the *MsgBox* as the argument. When this is done, the *MsgBox* object pops up and is activated. Because there is no other object specified in the argument to *Dyalog APL Language: Dq*, all other objects are de-activated. The only thing that the user can do (within the APL application) is to press one of the buttons in the *MsgBox*. When this happens, the *MsgBox* automatically pops down, the callback function (if any) is fired, and then *Dyalog APL Language: Dq* terminates.

Notice that the position and size of the *MsgBox* are determined by Windows and are fixed, although the *MsgBox* may be moved by the user after it has been displayed.

The `MsgBox` object generates one of three events; `MsgBtn1` (61), `MsgBtn2` (62), or `MsgBtn3` (63) depending upon which button is pressed.

```

Caption←'Default MsgBox' ♦ Text←'Hello World'
'Msg' □WC 'MsgBox' Caption Text ♦ □DQ 'Msg'

Caption←'Information MsgBox' ♦ Text←'Update Completed'
'Msg' □WC 'MsgBox' Caption Text 'Info' ♦ □DQ 'Msg'

Caption←'Query MsgBox' ♦ Text←'Save Changes'
'Msg' □WC 'MsgBox' Caption Text 'Query' ♦ □DQ 'Msg'

Caption←'Warning MsgBox'
Text←'Calculations will take 10 minutes'
'Msg' □WC 'MsgBox' Caption Text 'Warn' ♦ □DQ 'Msg'

Caption←'Error MsgBox'
Text←'Data out of range'
'Msg' □WC 'MsgBox' Caption Text 'Error' ♦ □DQ 'Msg'

Caption←'Custom MsgBox'
Text←'You can have a multi-line'
Text,←'message if you want one'
B←'ABORT' 'RETRY' 'IGNORE'
'Msg' □WC 'MsgBox' Caption Text 'Info' B ♦ □DQ 'Msg'

```

2.40 NetClient Object

Purpose: The `NetClient` object represents an instance of a Microsoft .NET class.

Parents NetType Root

Children Timer

Description

The `NetClient` object represents an instance of a .NET class.

Normally, you create a `NetClient` object using the `New` method. For example:

```

□USING ← 'System'
DT1←DateTime.New 2002 4 30
DT1.Type
NetClient

```

If, for any reason, you are unable to use the `New` method, you may create a `NetClient` object using `□WC`. In this case, the `ClassName` property specifies the *full* name of the .NET class, and the `ConstructorArgs` property specifies the arguments for the constructor function if required.

```

□USING ← 'System'
'DT2'□WC'NetClient' 'System.DateTime'(1949 4 30)
DT2.(Type ClassName ConstructorArgs)
NetClient System.DateTime 1949 4 30

```

2.41 NetControl Object

Purpose: The `NetControl` object provides the means to instantiate a .NET control in the Dyalog GUI.

Parents Form Grid Group PropertyPage SubForm

Children NetClient OLEClient Timer

Properties Type Posn Size Coord ClassName Attach MethodList ChildList EventList PropList

Description

In principle, you may use the `NetControl` to embed any class that derives from `System.Windows.Forms.Control` (from `system.windows.forms.dll`), including derived classes written in Dyalog APL.

To load a particular .NET control, the appropriate .NET Assembly must be specified in `□USING`; otherwise the expression will cause a `LIMIT ERROR`. For example, to load one of the standard .NET controls:

```

□USING, ← 'System.Windows.Forms,system.windows.forms.dll'

```

The [Section 6.97](#) property specifies the name of the .NET control to be instantiated and to which the new object named by the left argument of `□WC` is to be connected. [Section 6.97](#) may only be specified by `□WC`.

Once you have created an instance of a particular NetControl, the properties, events and methods it supports may be obtained using `□NL`. These are the properties, events and methods defined for the control by its author. The "Dialog" properties listed above, are not reported by `□NL`, but take precedence over (that is, mask) any members of the same name that may be exposed by the class itself.

The following example illustrates the use of the Button class. In this case, the FlatStyle property of the button is set to "Popup". This gives the button a flat appearance until the mouse is hovered over it, when its appearance it changes to 3-dimensional.

```

□USING←'System'
□USING,←c'System.Windows.Forms,system.windows.forms.dll'
□USING,←c'System.Drawing,system.drawing.dll'
an←□NEW FontFamily(c'Arial')
myfont←□NEW Font(an 24 FontStyle.Bold GraphicsUnit.Point)

'f'□WC'Form'('Coord' 'Pixel')('Size' 120 200)
f.Caption←'NetControl'
'f.l'□WC'Label' 'Button with FlatStyle=Popup'(2 2)

'f.b'□WC'NetControl' 'Button'('Size' 60 160)

f.b.□NL ~2
AutoSizeMode DialogResult AutoEllipsis AutoSize BackColor FlatStyle Flat
Appearance...

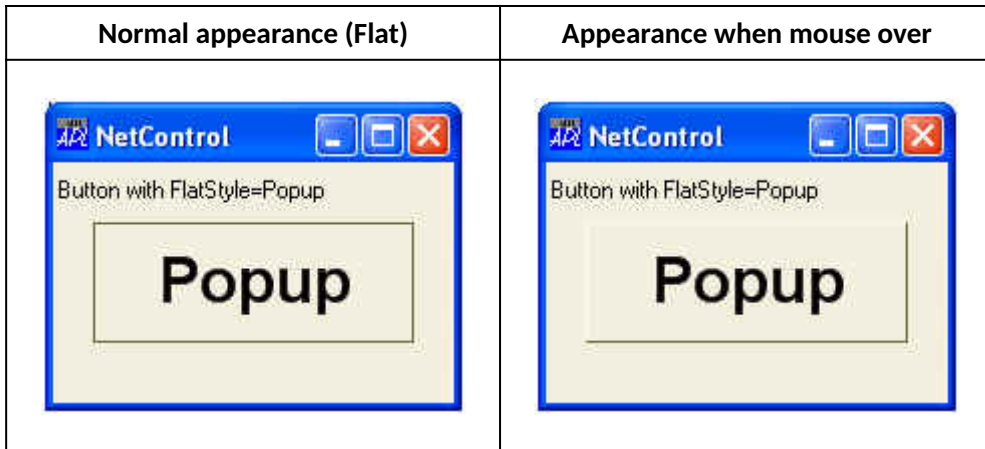
f.b.□NL ~3
BeginInvoke BringToFront Contains CreateControl CreateGraphics CreateObj
Ref Dispose DoDragDrop...

f.b.□NL~8
DoubleClick MouseDoubleClick AutoSizeChanged ImeModeChanged BackColorCha
nged...

f.b.Text←'Popup'
f.b.Font←myfont

f.b.(FlatStyle←FlatStyle.Popup)

```

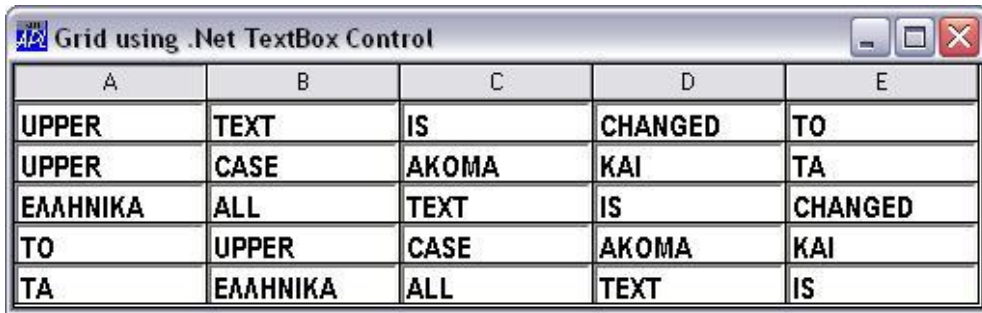


In most cases, you may use a NetControl in the cells of a [Section 6.263](#) object. Unless you specify otherwise, using the [Section 6.324](#) property of the [Section 6.263](#), the default property of the NetControl will be associated with the corresponding element of Values. The following example illustrates the use of a TextBox control. In this example, the CharacterCasing property of the TextBox is set to Upper, causing all text to be converted to upper-case.

```

[USING←'System'
[USING,←c'System.Windows.Forms,system.windows.forms.dll'
[USING,←c'System.Drawing,system.drawing.dll'
an←[NEW FontFamily(c'Arial Narrow')
myfont←[NEW Font(an 11 FontStyle.Bold GraphicsUnit.Point)
'f'[WC'Form'('Coord' 'Pixel')('Size' 130 500)
f.Caption←'Grid using .NET TextBox Control'
'f.g'[WC'Grid'('Posn' 0 0)f.Size
f.g.(ShowInput TitleWidth)←1 0
'f.g.tb'[WC'NetControl' 'TextBox'
f.g.tb.Font←myfont
f.g.tb.(CharacterCasing←CharacterCasing.Upper)
f.g.Input←'f.g.tb'
wds←'All' 'Text' 'Is' 'Changed' 'to' 'Upper' 'case'
wds,←'ακομα' 'kai' 'τα' 'Ελληνικά'
f.g.Values←5 5pws

```



A	B	C	D	E
UPPER	TEXT	IS	CHANGED	TO
UPPER	CASE	AKOMA	KAI	TA
EAHNIKA	ALL	TEXT	IS	CHANGED
TO	UPPER	CASE	AKOMA	KAI
TA	EAHNIKA	ALL	TEXT	IS

Implementation note: The instance of the .NET control is actually placed inside an instance of the .NET class `System.Windows.Forms.ContainerControl`. This `ContainerControl` is then embedded in the Dyalog parent, such as a [Section 6.227](#). This "extra level" should have no effect on how the control is used or on how it behaves.

2.42 NetType Object

Purpose: The `NetType` object is used to export a namespace as a Microsoft.NET class.

Parents Root

Children Bitmap NetClient TCPSocket Timer

Properties BaseClass

Methods

Events

Description

The `NetType` object allows you to export an APL namespace as a .NET class that can be accessed by any conforming .NET client application.

The `BaseClass` property specifies the name of the .NET class from which the specified `NetType` object inherits. The default is `System.Object`.

When you create a `NetType` object, the name of its parent namespace specifies the name of the corresponding Microsoft .NET Namespace to which the `NetType` class belongs. If the `NetType` is created as a child of root, the corresponding Microsoft .NET Namespace is unnamed.

2.43 OCXClass Object

Purpose: This object provides access to OLE (ActiveX) Controls.

Parents ActiveXControl CoolBand Form Grid OLEServer PropertyPage Root SubForm
TCPSocket ToolBar ToolControl

Properties Type ClassName Event Data Translate ClassID KeepOnClose TypeList HelpFile
ToolboxBitmap LicenseKey QueueEvents MethodList ChildList EventList PropList

Methods Detach GetPropertyInfo GetEventInfo GetMethodInfo GetTypeInfo
SetPropertyInfo SetMethodInfo ShowHelp ShowProperties Browse OLEAddEventSink
OLEDeleteEventSink OLEListEventSinks

Events

Description

This object loads an OLE Control into memory and defines a new class of object associated with it. The name of the new class is the name specified by the left argument of `WC`. You may create an instance of the newly defined class using the name you assigned to the OCXClass object as the Type property.

Note that you may not create an instance of OCXClass using `NEW`.

Once you have defined a new OCXClass, the properties, events and methods it supports may be obtained from its *PropList*, *EventList* and *MethodList* properties. These are the properties, events and methods defined for the OLE control by its author.

The *QueueEvents* property determines how events reported by the ActiveX control are handled.

To find out how to use the OLE control, you must consult the appropriate documentation. However, a great deal of information about it can be obtained using the *GetPropertyInfo*, *GetEventInfo*, and *GetMethodInfo* methods.

2.44 OLEClient Object

Purpose: The OLEClient object provides access to an OLE Automation Server

Parents ActiveXControl CoolBand Form OLEServer Root TCPSocket

Children Form TCPSocket Timer

Properties Type ClassName Event Data Handle ClassID KeepOnClose TypeList HelpFile
 LastError Locale AutoBrowse QueueEvents InstanceMode LateBind MethodList
 ChildList EventList PropList

Methods Browse Detach GetEventInfo GetMethodInfo GetPropertyInfo GetTypeInfo
 OLEAddEventSink OLEDeleteEventSink OLEListEventSinks OLEQueryInterface
 SetMethodInfo SetPropertyInfo ShowHelp

Events

Description

The OLEClient object allows you to control OLE Servers, which may be written in a variety of different programming languages, including Dyalog APL itself.

The *ClassName* property specifies the name of the OLE object to which the new object named by the left argument of `⎕WC` is to be connected. A list of all the OLE Server objects installed on your system may be obtained from the *OLEServers* property of *Root*. *ClassName* may only be specified by `⎕WC`.

Alternatively, the OLE object may be identified by the *ClassID* property.

The *AutoBrowse* property and *Browse* method are no longer relevant and are ignored. They are retained only for backwards compatibility with previous versions of Dyalog APL.

Note that the *PropList* and *MethodList* properties of an OLEClient instance contain the names of the properties and methods exposed by the corresponding OLE Object in addition to the generic properties and methods of the OLEClient class.

If you call an OLE method with an invalid parameter, set a read-only property, or assign it an invalid value, the *LastError* property of the OLEClient and *Root* objects will contain error information generated by OLE.

2.45 OLEServer Object

Purpose: The OLEServer object is used to establish a namespace as an OLE Server object that can be used by an OLE Automation client.

Parents ActiveXControl Form OLEServer Root

Children Bitmap BrowseBox Clipboard Cursor FileBox Font Form Icon ImageList Menu
 Metafile MsgBox OCXClass OLEClient OLEServer Printer PropertySheet TCPSocket Timer
 TipField

Properties Type ClassName Event Data Handle ExportedFns ExportedVars ClassID
 KeepOnClose TypeLibID TypeLibFile ServerVersion LastError RunMode ShowSession
 LateBind MethodList ChildList EventList PropList

Methods Detach OLERegister OLEUnregister SetFnInfo SetVarInfo SetEventInfo

Events Close Create

Description

The OLEServer object allows you to export an APL namespace so that its functions and variables become directly accessible to an OLE Automation client application such as Microsoft Visual Basic or Microsoft Excel.

An OLEServer may be saved as an *out-of-process* OLE server (in a workspace) or as an *in-process* OLE server (in a DLL). See *Interface Guide* for details.

When you create an OLEServer object, APL allocates various OLE attributes to it. For example, the CLSID, which uniquely identifies the object, is assigned at this stage. However, the object is not actually *registered* until you execute)SAVE.

Registration involves updating the Windows registry with information about the object itself, such as its name, the command required to obtain it and so forth. Registration also records information about all of the functions and variables that your object exposes. Registration is therefore a non-trivial operation and should be delayed until the point when you are ready to test your OLEServer.

You may create an empty OLEServer object and then define functions and variables within it. Alternatively, you may convert an existing namespace which is already populated with functions and variables. The latter method is recommended as it implies less registry activity during the development of the object.

The *ExportedFns* and *ExportedVars* properties specify the names of the functions and variables that will be exposed by the object to OLE clients.

The *RunMode* property is a character vector that specifies how the object serves multiple clients. It may be 'MultiUse' (the default), 'SingleUse', or 'RunningObject'.

The *ShowSession* property is either 0 (the default) or 1 and specifies whether or not the APL Session window is displayed when the first instance of the OLEServer is created.

RunMode and *ShowSession* apply only to *out-of-process* OLEServers.

2.46 Poly Object

Purpose: A graphical object used to draw lines, polygons, and filled areas.

Parents ActiveXControl Animation Bitmap Button ButtonEdit Combo ComboEx Edit Form Grid Group Label List ListView MDIClient Metafile Printer ProgressBar PropertyPage PropertySheet RichEdit Scroll Spinner Static StatusBar SubForm TabBar TipField ToolBar TrackBar TreeView UpDown

Children Timer

Properties Type Points FCol BCol LStyle LWidth FStyle FillCol Coord Visible Event Dragable OnTop CursorObj AutoConf Data Accelerator KeepOnClose DrawMode MethodList ChildList EventList PropList

Methods Detach

Events Close Create DragDrop MouseDown MouseUp MouseMove MouseDbClick Help Select

Description

The *Points* property specifies one or more sets of co-ordinates through which one or more lines are drawn. The resulting polygon(s) may also be filled.

LStyle and *LWidth* define the style and width of the lines. *FCol* and *BCol* determine the colour of the lines.

FStyle specifies whether or not the polygon(s) are filled, and if so, how. For a solid fill (*FStyle* 0), *FillCol* defines the fill colour used. For a pattern fill (*FStyle* 1-6) *FillCol* defines the colour of the hatch lines and *BCol* the colour of the areas between them.

Note that if you specify filling, you do not have to define a **closed** polygon. The first and last points will automatically be joined for you if necessary.

The value of *Dragable* determines whether or not the object can be dragged. The value of *AutoConf* determines whether or not the Poly object is resized when its parent is resized.

The structure of the property values is best considered separately for single and multiple polylines or polygons.

Single Polyline or Polygon

For a single polyline or polygon, *Points* is either a 2-column matrix of (y,x) co-ordinates, or a 2-element vector of y and x co-ordinates respectively.

LStyle and *LWidth* are both simple scalar numbers.

FStyle is either a single number specifying a standard fill pattern, or the name of a *Bitmap* object which is to be used as a "brush" to fill the polygon.

FCol, *BCol* and *FillCol* are each either single numbers representing standard colours, or 3-element vectors which specify colours explicitly in terms of their RGB values.

EXAMPLES

First make a *Form* :

```
'F' □WC 'Form'
```

Draw a single line from (y=20, x=10) to (y=30, x=50)

```
'F.L1' □WC 'Poly' ((20 30)(10 50))
```

or

```
L ← 2 2p20 10 30 50
'F.L1' □WC 'Poly' L
```

Draw a horizontal line from (y=20, x=10) to (y=20, x=50). Note scalar extension of y-coordinate.

```
'F.L1' □WC 'Poly' (20(10 50))
```

Draw an empty box in green :

```
Y ← 10 10 50 50 10
X ← 10 50 50 10 10
'F.L1' □WC 'Poly' (Y X) (0 255 0)
```

Ditto, using a green/blue dashed line (*LStyle* 1) :

```
'F.L1' □WC 'Poly' (Y X) (0 255 0)(0 0 255) 1
```

Draw a red filled rectangle with a black border 5 pixels wide :

```
'F.L1' WC 'Poly' (Y X) (0 0 0) ('LWidth' 5)
('FStyle' 0)('FillCol' 255 0 0)
```

Multiple Polylines/Polygons

To draw a set of polylines or polygons with a single name, *Points* is a nested vector whose items are themselves 2-column matrices or 2-element nested vectors.

LStyle and *LWidth* may each be simple scalar values (applying to all the polylines) or simple vectors whose elements refer to each of the corresponding polylines in turn.

FStyle may be a simple scalar numeric or a simple character vector (*Bitmap* name) applying to all polylines, or a vector whose elements refer to each of the corresponding polylines in turn.

Similarly, *FCol*, *BCol* and *FillCol* may each be single numbers or a single (enclosed) 3-element vector applying to all the polylines. Alternatively, these properties may contain vectors whose elements refer to each of the polylines in turn. If so, their elements may be single numbers or nested RGB triplets, or a combination of the two.

EXAMPLES

First make a *Form* :

```
'F' WC 'Form'
```

Draw two concentric triangles :

```
BY ← 10 10 50 10
BX ← 15 65 40 15
RY ← 15 15 40 15
RX ← 25 55 40 25
'F.L1' WC 'Poly' ((BY BX)(RY RX))
```

Or, using matrices :

```
BT ← BY,[1.5]BX
RT ← RY,[1.5]RX
'F.L1' WC P←'Poly' (BT RT)
```

Ditto, but draw the first blue, the second red :

```
'F.L1' WC P,c((0 0 255)(255 0 0))
```

Ditto, but make the lines 3 pixels wide :

```
'F.L1' □WC P, ((0 0 255)(255 0 0))('LWidth' 3)
```

Ditto, but make the line widths 3 and 6 pixels respectively :

```
'F.L1' □WC P, ((0 0 255)(255 0 0))('LWidth' 3 6)
```

Draw the first hollow, but fill the second in green :

```
'F.L1' □WC P, ('FStyle' -1 0)('FillCol' (c0 255 0))
```

2.47 Printer Object

Purpose: To provide printer output.

Parents ActiveXControl CoolBand Form OLEServer PropertyPage PropertySheet Root TCPSocket

Children Bitmap Circle Ellipse Font Icon Image Marker Metafile Poly Rect Text Timer

Properties Type PName DevCaps Coord Event FontObj FontList YRange XRange Data TextSize EdgeStyle Handle Orientation Copies PrintRange Collate PaperSize PaperSizes PaperSource PaperSources ColorMode Resolution Resolutions Duplex Translate Accelerator KeepOnClose MethodList ChildList EventList PropList

Methods Detach Print Setup NewPage Abort RTFPrintSetup GetTextSize

Events Close Create Select

Description

The *PName* property is a character vector which specifies the name of an installed printer and the device to which it is attached. The name and device are separated by a comma (,). All valid values of *PName* can be obtained from the *PrintList* property of the *Root* object.

If not specified, the default value of *PName* is (>'.' □WG 'PrintList').

The *DevCaps* property reports the size of the printable area of the page in pixels (dots) and in millimetres. It also reports the number of colours available. This is 2 on a monochrome printer (black and white), although grey scales may be available.

The *FontList* property provides a list of fonts that are applicable and includes TrueType and printer fonts. This list is typically different from that obtained from the *FontList* property on the *Root* object which lists those fonts that apply to the screen.

The Orientation property specifies the orientation of the page and may be either 'Portrait' or 'Landscape'.

The graphics objects listed above may be printed in much the same way as they may be displayed on a *Form* or *Static*. The differences are :

Once an object has been created, it will be printed, even if its name is subsequently expunged.

An object does not **replace** an existing one which has the same name.

The act of changing one or more properties of a named object causes the object to be printed a second time. For example, changing the *Posn* of an object will print it again at a different place.

In general it is recommended that you use unnamed objects for printing.

The Printer object five methods :

Name	Event	Description
<i>Print</i>	100	Sends output to print spooler
<i>Setup</i>	101	Displays Printer Set-up dialog box
<i>NewPage</i>	102	Throws a new page
<i>Abort</i>	103	Aborts the print job
<i>RTFPrintSetup</i>	460	Displays Printer Set-up dialog box

EXAMPLES

Start a print job on the default printer

```
'PR1' □WC 'Printer'
```

Write a centred heading at the top of the page using a proportional font

```
'PR1.' □WC 'Text' 'Report Title' (0 50)('HAlign' 1) ('FontObj'
'Roman' 64)
```

Draw a line across the page, 2 pixels wide

```
'PR1.' □WC 'Poly' (2(0 100)) ('LWidth' 2)
```

Print a character matrix. Note that a fixed width font is used.

```
REPORT ← 'I6' □FMT ?20 6p1000
'PR1.' □WC 'Text' REPORT (10 0)('FontObj' 'DyalogAPL')
```

Throw a new page

```
PR1.NewPage
```

Spool output

```
□EX 'PR1'
```

2.48 ProgressBar Object

Purpose: The ProgressBar object is used to indicate the progress of a lengthy operation.

Parents ActiveXControl CoolBand Form Group PropertyPage StatusBar SubForm
ToolBar ToolControl

Children Bitmap Circle Cursor Ellipse Font Icon Marker Poly Rect Text Timer

Properties Type Posn Size Style Coord Active Visible Event Thumb Step Wrap Limits
Sizeable Dragable BCol CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj
Tip TipObj Translate Accelerator AcceptFiles KeepOnClose ProgressStyle Redraw
TabIndex Interval MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ProgressStep

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose
Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick
MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate
GestureTwoFingerTap GesturePressAndTap Select

Description

The ProgressBar object is a window that an application can use to indicate the progress of a lengthy operation. The appearance of the bar in the ProgressBar is determined by the *ProgressStyle* property.

If *ProgressStyle* is *Normal* or *Smooth*, the size of the bar, intended to indicate the amount of progress, is determined using the *Thumb* property in relation to its *Limits* property, and/or using the *ProgressStep* method. This can be updated as appropriate in the application logic or by using a *Timer*.

The range of a `ProgressBar` is specified by the *Limits* property. This is a 2-element integer vector defining its minimum and maximum values. The position of the filled rectangle is specified by the *Thumb* property. You can update the `ProgressBar` by using `ProgressBar.SetValue` to set the value of the *Thumb* directly, or by using the *ProgressStep* method. The latter causes the *Thumb* to be updated by the value of the *Step* property.

If you attempt to set the *Thumb* to a value greater than its maximum value (using either method) the behaviour depends upon the value of the *Wrap* property which is Boolean and has a default value of 1. If *Wrap* is 1, the value obtained when you set the *Thumb* property is given by the expression:

$$\text{LIMITS}[1] + (1 + \text{LIMITS}[2] - \text{LIMITS}[1]) * (\text{THUMB} - \text{LIMITS}[1])$$

where `THUMB` is the value to which you set the *Thumb* property and `LIMITS` is the value of the *Limits* property. This causes the highlighted rectangle to begin filling again from the left.

If *ProgressStyle* is *Marquee*, the size of the bar is fixed and its position changes with time according to the value of the *Interval* property. The values of *Thumb*, *Limits*, *Wrap* and *Step* are irrelevant.

2.49 PropertyPage Object

Purpose: The `PropertyPage` object represents a single page in a `PropertySheet`.

Parents `PropertySheet`

Children `Animation` `Bitmap` `BrowseBox` `Button` `ButtonEdit` `Calendar` `Circle` `Clipboard` `ColorButton` `Combo` `ComboEx` `Cursor` `DateTimePicker` `Edit` `Ellipse` `FileBox` `Font` `Grid` `Group` `HTMLRenderer` `Icon` `Image` `ImageList` `Label` `List` `ListView` `Locator` `Marker` `Metafile` `MsgBox` `OCXClass` `Poly` `Printer` `ProgressBar` `Rect` `RichEdit` `Scroll` `SM` `Spinner` `Splitter` `Static` `SubForm` `TCPsocket` `Text` `Timer` `TipField` `TrackBar` `TreeView` `UpDown`

Properties `Type` `Caption` `Posn` `Size` `Coord` `Active` `Event` `HasHelp` `FontObj` `Data` `EdgeStyle` `Hint` `HintObj` `Tip` `TipObj` `Changed` `Translate` `AcceptFiles` `KeepOnClose` `MethodList` `ChildList` `EventList` `PropList`

Methods `Detach` `ChooseFont` `GetTextSize` `Animate` `GetFocus` `ShowSIP` `GetFocusObj`

Events `Close` `Create` `DragDrop` `Configure` `ContextMenu` `DropFiles` `DropObjects` `Expose` `Help` `KeyPress` `GotFocus` `LostFocus` `MouseDown` `MouseUp` `MouseMove` `MouseDoubleClick` `MouseEnter` `MouseLeave` `MouseWheel` `GestureZoom` `GesturePan` `GestureRotate`

GestureTwoFingerTap GesturePressAndTap FontOK FontCancel PageApply PageCancel
PageHelp PageBack PageNext PageFinish PageChanged PageActivate PageDeactivate

Description

The *PropertyPage* object represents a single page within a *PropertySheet*.

The *Posn* and *Size* properties are read-only properties determined by the parent *PropertySheet* and may not be changed using `WC` or `WS`.

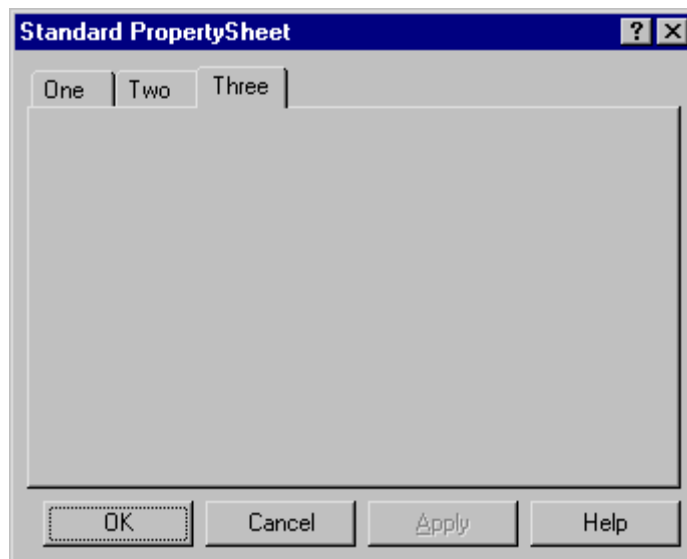
The *HasHelp* property is either 1 (the default) or 0. If the parent *PropertySheet* has a "Help" button (determined by its own *HasHelp* property) this property determines whether or not the Help button is active when the *PropertyPage* is the current page. If the *HasHelp* property of a *PropertyPage* is 0, the Help button on the parent *PropertySheet* will be temporarily disabled when that *PropertyPage* is displayed.

The *PropertyPage* object generates a *PageActivate* event when it becomes the current page and a *PageDeactivate* event when another page is selected. These events may not be disabled by a callback function.

If the user presses the Cancel button, the current *PropertyPage* generates a *PageCancel* event. This is followed by a Close event which is reported by the parent *PropertySheet*.

Other properties and behaviour depend upon the Style of the parent *PropertySheet* which may be 'Standard' or 'Wizard'

Standard Behaviour



In a *Standard* *PropertySheet*, the *Caption* property of each *PropertyPage* specifies the text that is written in its tab.

PropertyPage objects owned by a *Standard PropertySheet* generate *PageCancel*, *PageApply* and *PageHelp* events. These events are all caused by the user pressing the corresponding button in the parent *PropertySheet*.

Conventionally, the *Apply* button is initially inactive. When the user changes an item on any of the *PropertyPages*, the *Apply* button immediately becomes active. When the user clicks the *Apply* button, the application responds (normally by changing the appropriate properties) and then the *Apply* button becomes inactive once again. This process is controlled as follows.

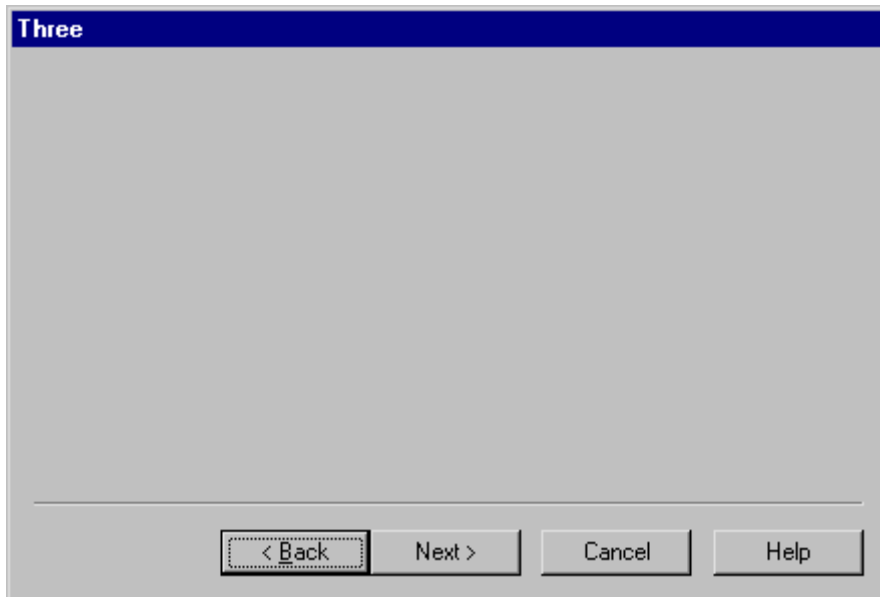
The *Changed* property is a Boolean value that determines whether or not a *PropertyPage* is marked as having been in any way altered. The *Apply* button is active if the value of the *Changed* property for *any* of the *PropertyPages* is 1, and is inactive otherwise

Initially, the value of the *Changed* property for all of the *PropertyPages* is 0 and the *Apply* button is therefore inactive. If the user alters a control on a *PropertyPage*, by, for example typing into an *Edit* object or changing the *State* of a *Radio Button*, the *PropertyPage* immediately generates a *PageChanged* event with the parameter 1. The default processing for this event is to set the *Changed* property of the *PropertySheet* (to 1). This in turn activates the "Apply" button. If you return 0 from a callback on the *PageChanged* event, the *Changed* property remains 0 and the *Apply* button remains inactive.

When the user clicks the *Apply* button, each of the *PropertyPages* whose *Changed* flag is currently set to 1 generates a *PageApply* event. The default processing for this event is to generate a *PageChanged* event with the parameter 0. This in turn resets the *Changed* property of the *PropertyPage* to 0. Once all of the *Changed* flags have been reset, the *Apply* button becomes inactive. If you return 0 from a callback on any of the *PageChanged* events, the *Changed* property for the corresponding *PropertyPage* remains 1 and the *Apply* button remains active.

You may control the value of the *Changed* property using `PropertySheet::PageChanged` or by calling *PageChanged* as a method. In all cases, the *Apply* button is active if the value of *Changed* on any *PropertyPage* is 1, and inactive otherwise.

Wizard Behaviour



If the *PropertyPage* is owned by a *Wizard PropertySheet*, its *Caption* property specifies the text that appears in the title bar of the *PropertySheet* window when the *PropertyPage* is the current page. Note that a *Wizard PropertySheet* ignores its own *Caption* property.

There are effectively 3 page changing buttons on a *Wizard PropertySheet*, named *Back*, *Next* and *Finish*. The *Next* and *Finish* buttons actually occupy the same position and are mutually exclusive. The captions on the buttons are language-dependent.

Conventionally, the buttons change according to which of the *PropertyPages* is currently displayed. If the first one is displayed, the *Next* button is active but the *Back* button is inactive. When a middle page is displayed, both the *Next* and *Back* buttons are active. When the last page is displayed, the caption on the *Next* button changes to *Finish*. However, in some applications, the *Back* button may be disabled to prevent the user returning to a previous page.

When the user clicks the *Back* or *Next* button, the *PropertyPage* generates a *PageBack* or *PageNext* event followed by a *PageDeactivate* event. The new *PropertyPage* then generates a *PageActivate* event. These are followed by a *SetWizard* event which is generated by the parent *PropertySheet* and actually controls the state of the buttons. When the user clicks the *Finish* button, the *PropertyPage* generates a *PageFinish* event alone. All of these events reported by the *PropertyPage* are reported for information

only. Returning 0 from a callback function has no effect. You may however control the buttons using the *SetWizard* event.

2.50 PropertySheet Object

Purpose: The PropertySheet object represents a standard multi-page dialog box.

Parents ActiveXControl Form OLEServer Root SubForm TCPSocket

Children Bitmap BrowseBox Circle Clipboard Cursor Ellipse FileBox Font Icon Locator Marker Metafile MsgBox Poly Printer PropertyPage Rect Text Timer TipField

Properties Type Caption Posn Size Style Coord Active Visible Event HasApply HasHelp PageActive PageActiveObject HelpButton FontObj OnTop Data EdgeStyle Handle Hint HintObj Tip TipObj Translate KeepOnClose MethodList ChildList EventList PropList

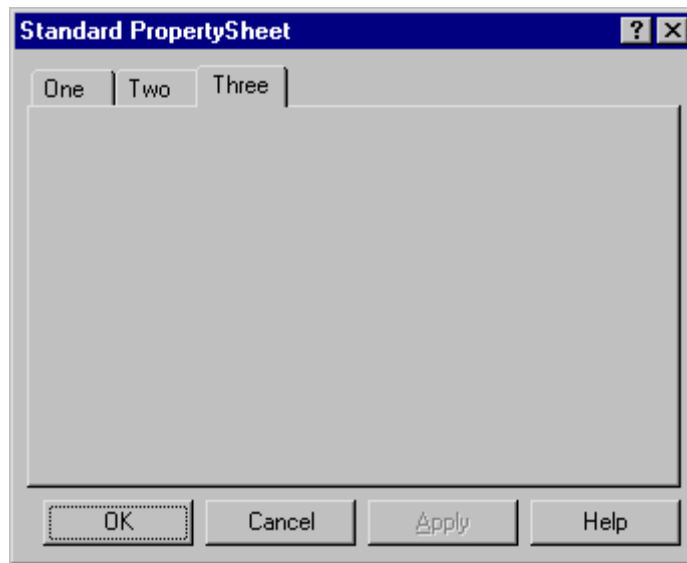
Methods Detach ChooseFont SetWizard SetFinishText CancelToClose GetFocus GetFocusObj Wait

Events Close Create FontOK FontCancel

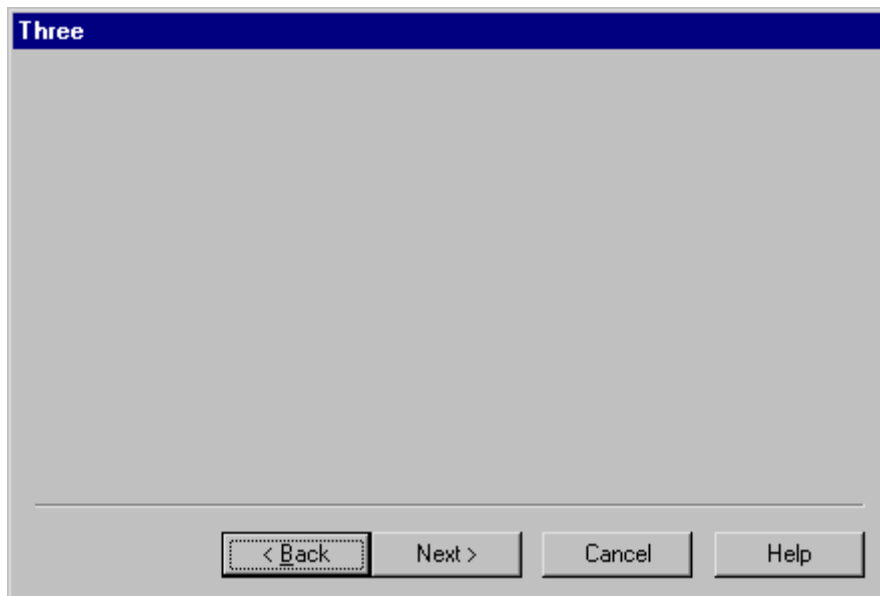
Description

There are two different kinds of PropertySheet which you select using the *Style* property. This may only be set when the PropertySheet is created using `□WC` and *Style* may not subsequently be changed using `□WS`.

If *Style* is *Standard* (the default), the PropertySheet displays a set of pages (each represented by a *PropertyPage*) as a set of tabbed forms as illustrated below. The user selects the current page by clicking on the appropriate tab. This *Style* allows the user to select any page at any time and does not oblige the user to visit any but the first page you choose to display. This *Style* is useful for displaying groups of options or settings that the user may change.



If Style is Wizard, the PropertySheet displays its pages in succession starting with the first. The user steps from one to another using the Next and Back buttons and may be forced to visit all the pages in a prescribed order. This Style is useful for data entry or for asking the user to make a series of choices.



The Caption property specifies the text written in the window title bar, but only applies if the Style is Standard. The title bar text of a Wizard PropertySheet is specified by the Caption of the current *PropertyPage*.

The `HasApply` and `HasHelp` properties are Boolean and specify whether or not the `PropertySheet` has "Apply" and "Help" buttons respectively. These properties may only be set when the object is created using `WC`. They both have default values of 1.

The `FontObj` and `EdgeStyle` properties have no effect on the appearance of the `PropertySheet` itself, but may be used to define the default appearance of its children.

2.51 Rect Object

Purpose: A graphical object used to draw boxes.

Parents `ActiveXControl` `Animation` `Bitmap` `Button` `ButtonEdit` `Combo` `ComboEx` `Edit` `Form` `Grid` `Group` `Label` `List` `ListView` `MDIClient` `Metafile` `Printer` `ProgressBar` `PropertyPage` `PropertySheet` `RichEdit` `Scroll` `Spinner` `Static` `StatusBar` `SubForm` `TabBar` `TipField` `ToolBar` `TrackBar` `TreeView` `UpDown`

Children `Timer`

Properties `Type` `Points` `Size` `Radius` `FCol` `BCol` `LStyle` `LWidth` `FStyle` `FillCol` `Coord` `Visible` `Event` `Dragable` `OnTop` `CursorObj` `AutoConf` `Data` `EdgeStyle` `Accelerator` `KeepOnClose` `DrawMode` `MethodList` `ChildList` `EventList` `PropList`

Methods `Detach`

Events `Close` `Create` `DragDrop` `MouseDown` `MouseUp` `MouseMove` `MouseDbClick` `Help` `Select`

Description

The *Points* property specifies one or more sets of co-ordinates which define the position(s) of one or more rectangles. The position of a rectangle is defined to be the position of the corner that is **nearest** to the origin of its parent. The default is therefore its top-left corner. The *Size* property specifies the height and width of each rectangle, measuring away from the origin.

The *Radius* property specifies the curvature of the corners of the rectangle.

LStyle and *LWidth* define the style and width of the lines used to draw the boundaries of the rectangle(s). *FCol* and *BCol* determine the colour of the lines.

FStyle specifies whether or not the rectangle(s) are filled, and if so, how. For a solid fill (*FStyle* 0), *FillCol* defines the fill colour used. For a pattern fill (*FStyle* 1-6) *FillCol* defines the colour of the hatch lines and *BCol* the colour of the spaces between them.

The *EdgeStyle* property may specify a 3-dimensional effect. If so, the boundary line around the rectangle is replaced by a border designed to achieve the desired effect.

The value of *Dragable* determines whether or not the object can be dragged. The value of *AutoConf* determines whether or not the Rect object is resized when its parent is resized.

The structure of the property values is best considered separately for single and multiple rectangles :

Single Rectangle

For a single rectangle, *Points* is either a 2-column matrix of (y,x) co-ordinates, or a 2-element vector of y and x co-ordinates respectively.

Size is a simple 2-element vector whose elements specify the height and width of the rectangle respectively.

Radius is a 2-element vector which specifies the major (y-axis) and minor (x-axis) radii of an ellipse used to draw the corners of the rectangle. Its default value is (0 0) which yields right-angled corners.

LStyle and *LWidth* are both simple scalar numbers.

FStyle is either a single number specifying a standard fill pattern, or the name of a *Bitmap* object which is to be used as a "brush" to fill the rectangle.

FCol, *BCol* and *FillCol* are each either single numbers representing standard colours, or 3-element vectors which specify colours explicitly in terms of their RGB values.

EXAMPLES

First make a *Form* :

```
'F' □WC 'Form'
```

Draw a single rectangle at (y=10, x=5) with height=30, width=50 :

```
'F.R1' □WC 'Rect' (10 5)(30 50)
```

Ditto with rounded corners (radii 10) :

```
'F.R1' □WC RC←'Rect' (10 5)(30 50)(10 10)
```

Ditto, but use a red line :

```
'F.R1' □WC RC, c(255 0 0)
```

Ditto, but fill in green

```
'F.R1' □WC RC(255 0 0) ('FStyle' 0)(0 255 0)
```

Multiple Rectangles

To draw a set of rectangles with a single name, *Points* may be a simple 2-element vector (specifying the location of all the rectangles), **or** a 2-column matrix whose first column specifies their y-coordinates and whose second column specifies their x-coordinates, **or** a 2-element nested vector whose first element specifies their y-coordinate(s) and whose second element specifies their x-coordinate(s).

Likewise, *Size* may be a simple 2-element vector (applying to all the rectangles), **or** a 2-column matrix whose first column specifies their heights and whose second column specifies their widths, **or** a 2-element nested vector whose first element specifies their height(s) and whose second element specifies their width(s).

Radius may be a simple 2-element vector (applying to all the rectangles), **or** a 2-column matrix whose first column specifies major radii and whose second column specifies minor radii, **or** a 2-element nested vector whose first element specifies major radii and whose second element specifies minor radii.

LStyle and *LWidth* may each be simple scalar values (applying to all the rectangles) or simple vectors whose elements refer to each of the corresponding rectangles in turn.

FStyle may be a simple scalar numeric or a simple character vector (*Bitmap* name) applying to all rectangles, or a vector whose elements refer to each of the corresponding rectangles in turn.

Similarly, *FCol*, *BCol* and *FillCol* may each be single numbers or a single (enclosed) 3-element vector applying to all the rectangles. Alternatively, these properties may contain vectors whose elements refer to each of the rectangles in turn. If so, their elements may be single numbers or nested RGB triplets, or a combination of the two.

EXAMPLES

First make a *Form* :

```
'F' □WC 'Form'
```

Draw two rectangles at (y=5, x=10) and (y=5, x=60) each of (height=40, width=10)

```
'F.R1' □WC 'Rect' ((5 5)(10 60)) (40 10)
```

Ditto, using scalar extension for (y=5) :

```
'F.R1' □WC 'Rect' (5(10 60)) (40 10)
```

Ditto, but draw the first with (height=40, width=30) and the second with (height=20, width=10) :

```
'F.R1' □WC 'Rect' (5(10 60)) ((40 20)(30 10))
```

Draw two rectangles at (y=5, x=10) and (y=5, x=60) each of (height=40, width=10) and with rounded corners of radii (10,10) :

```
'F.R1' □WC RC+ 'Rect' (5(10 60)) (40 10) (10 10)
```

Ditto, using a green line for both :

```
'F.R1' □WC RC, <=0 255 0
```

Ditto, but using red and blue lines respectively :

```
'F.R1' □WC RC, c(255 0 0)(0 0 255)
```

2.52 RichEdit Object

Purpose: The RichEdit object is a multi-line text editor that provides a wide range of word-processing capabilities.

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm ToolBar ToolControl

Children Bitmap Circle Cursor Ellipse Font Icon Marker Poly Rect Text Timer

Properties Type Text Posn Size File Coord Border Active Visible Event VScroll HScroll SelText Sizeable Dragable FontObj FCol BCol CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Changed RTFText Translate Accelerator CharFormat WordFormat ParaFormat PageWidth AcceptFiles WantsReturn KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj RTFPrintSetup RTFPrint FileRead FileWrite

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp

MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom
 GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap Protected
 Change Select

Description

A RichEdit object is a window in which the user can enter and edit text. The text can be assigned character and paragraph formatting. It is implemented using the Microsoft Windows RichEdit Version 1.0 control, although embedded OLE objects are not supported.

The RichEdit object provides a programming interface for formatting text. However, your application must implement any user interface components necessary to make formatting operations available to the user. For example, your program can set the colour and font of a particular block of text, but the RichEdit itself provides no facilities for the user to do this directly. It is up to you to provide these.

The *File* property specifies the name of a file associated with the object. Data in the file is assumed to be in rich text format, and the default extension for the file is .RTF. You can read the file into the object by calling *FileRead* and you can write the contents to the file by calling *FileWrite*. You can also print the contents of the object by calling *RTFPrint*.

The *Text* property may be used to set or retrieve the text of the RichEdit, but ignores formatting information. *Text* may set using a simple character vector, a simple matrix, or a vector of vectors. If *Text* is specified by a matrix or by a vector of vectors, "new-line" characters are automatically added at the end of each line in the RichEdit control.

The user may insert a "new-line" character in the text by pressing Ctrl-Enter. If *Text* was set by a matrix, it is returned as a matrix. Otherwise it is returned as a vector of vectors. "New-line" characters are not returned. If *Text* was not specified by *Dyalog APL Language: Wc* or *Dyalog APL Language: Wsit* it is returned an empty matrix (1 0ρ ' ').

The *RTFText* property may be used to set or retrieve the contents of the RichEdit, including text and formatting.

The *PageWidth* property defines the width of the text within the object. Text entered into the object is automatically wrapped according to *PageWidth*. This property also defines the width when the text is printed.

You can set the default character format or the format of a particular block of text using the *CharFormat* property. If there is no selection, setting *CharFormat* defines the default character format that applies at the current insertion position and establishes the appearance of all of the text (font, colour, size etc.) that the user subsequently

enters here. If there is a selection, setting *CharFormat* sets the character format for the selected block of text.

The *WordFormat* property is similar to *CharFormat* except that it sets the format for the selected word(s) or, if there is no selection, for the word containing the insertion point.

The *ParaFormat* property defines the paragraph formatting which includes alignment, indentation and the location of tab stops. When you set *ParaFormat* with ¶WS, the formatting is applied to the current selection. If there is no selection, it defines the default paragraph formatting at the insertion point.

All of the dimensions used for text and paragraph formatting are specified in *Twips*. You can convert from pixels to *Twips* and vice versa using the *DevCaps* property of either *Root* or the *Printer* object as appropriate.

The behaviour of the Enter key is defined by the *WantsReturn* property. If *WantsReturn* is 1 (the default), the Enter key inputs a new line into the RichEdit object. If *WantsReturn* is 0 the Enter key is ignored by the RichEdit object and may instead generate a Select event on a Button. In this case the user must press Ctrl+Enter to input a new line.

The user may copy and paste information (in RTF format) between a RichEdit object and the Windows clipboard. The Clipboard object also has an *RTFText* property that supports RTF format

If the user attempts to alter text that is protected (see *CharFormat*) the RichEdit object reports a *Protected* event.

You may print the contents of a RichEdit object using the *RTFPrint* method. You may display a print set-up dialog box using the *RTFPrintSetup* method.

2.53 Root Object

Purpose: This is an invisible "system" object that acts as the parent of all other objects.

Children Bitmap BrowseBox Clipboard Cursor FileBox Font Form HTMLRenderer Icon ImageList Locator Menu Metafile MsgBox NetClient NetType OCXClass OLEClient OLEServer Printer PropertySheet SysTrayItem TCPSocket Timer TipField

Properties Type Caption Posn Size DevCaps Coord Event FontObj FontList PrintList IconObj CursorObj YRange XRange Data TextSize Yield EdgeStyle HintObj TipObj

Translate APLVersion EvaluationDays KeepOnClose OLEControls OLEServers LastError
RadiusMode MethodList ChildList EventList PropList

Methods ChooseFont Flush NameFromHandle GreetBitmap GetCommandLine
GetCommandLineArgs GetEnvironment ListTypeLibs DeleteTypeLib SetServiceState
GetServiceState GetTextSize GetFocus ShowSIP GetFocusObj IDNToDate DateToIDN
TCPGetHostID GetBuildID Wait

Events FontOK FontCancel DDE Idle ExitWindows ExitApp WinIniChange
SysColorChange DisplayChange ActivateApp ServiceNotification

Description

There is a single Root object called `'.'` which is always present. It cannot be created using *Dyalog APL Language: Wc* nor can it be destroyed.

The *Caption* and *IconObj* properties of `'.'` are used to identify a Dyalog APL/W application as distinct from the APL Session. The *Caption* property specifies the application name that is displayed when you cycle through running applications using Alt+Tab and by the Windows Task List. The *IconObj* property specifies the name of an *Icon* object that is displayed alongside the application name in the box displayed by Alt+Tab. For these to take effect, your application must have at least one visible and active *Form*.

For the Root object, the value of *Posn* is (0,0). The value of *Size* is either (100,100) if *Coord* is `'Prop'`, or the size of the screen in pixels if *Coord* is `'Pixel'`. *XRange* and *YRange* both have the value (0,100). The *DevCaps* property reports the physical size of the screen in terms of both pixels and millimetres. It also reports the number of colours available.

The *FontList* property provides a list of all the character fonts that are available. The *PrintList* property provides a list of all the installed printers. These properties are *read-only* and may not be changed using *Dyalog APL Language: Ws*

As the default value of *Coord* is `'Inherit'` for all other objects, the value of *Coord* for `'.'` defines the default co-ordinate system. It may be either `'Prop'` (the default) or `'Pixel'`. `'Inherit'` and `'User'` are not allowed.

The *CursorObj* property is used to define a cursor for the application as a whole. Its default value is an empty character vector. If it is set to any value other than `' '` or 0, the selected cursor overrides the *CursorObj* values for all other objects. If you want to indicate that the application is "busy", you can therefore set the *CursorObj* property on `'.'` to an hourglass for the duration of the operation. For example:

```
'.' □WS 'CursorObj' 1  ⌘ Set cursor to an hourglass
```

[lengthy process...]

```
'.' □WS 'CursorObj' 0  ⌘ Reset cursor
```

The *Yield* property specifies how frequently APL yields to Windows during the execution of code. Its default value is 200 milliseconds.

The *EdgeStyle* property is used to determine whether or not objects may have 3-dimensional effects. Setting *EdgeStyle* to 'None' disables 3-dimensional effects on all *Forms* and controls. Setting *EdgeStyle* to any other value enables 3-dimensional effects for these objects.

The *ExitApp* and *ExitWindows* events can be used to prevent the user closing your application from the Windows Task List or by terminating Windows.

The expression □EX ' .' deletes all objects owned by the current thread **except** the Root object itself. In addition, if this expression is executed by thread 0, it resets all the properties of ' .' to their default values.

Exposing Root members

The Properties, Methods and Events of the Root object are always accessible using the system functions □WS, □WG and □NQ but may also be optionally accessed directly as if they were global variables or functions in the workspace.

For example, if Root members are exposed, the following expression will set the application cursor (GUI) to an hourglass:

```
CursorObj←1
```

There are a number of elements that control whether or not Root members are exposed.

1. The fundamental mechanism is a flag that is saved in every workspace. If this flag is set, the members of the Root object are exposed; if not, they are not exposed.
2. This flag may be changed dynamically using the *Options/Object Syntax/Expose Root Properties* menu item on the Session or using 2401⌘. If the workspace is subsequently saved, the current value of the flag is saved with it.
3. The value of the flag in a CLEAR WS is determined by the **PropertyExposeRoot** parameter. Under Windows, this parameter is associated with the *Expose properties of Root* checkbox on the *Object Syntax* Tab of the *Configuration* dialog

box. When you change the value of this checkbox and close the *Configuration* dialog by clicking *OK*, the value of the **PropertyExposeRoot** parameter is immediately updated in the user's section of the Registry. However, the value of the flag *in the current workspace* is not changed. The **PropertyExposeRoot** parameter only defines the value of the flag in a `CLEAR WS`, so if you subsequently type `)CLEAR`, the current value of the parameter in the Registry determines whether or not Root members are exposed and sets the flag in the workspace accordingly.

For further information, see *User Interface: Session Menubar, Installation/Configuration: Configuration Parameters*, and *Dyalog APL Language: Expose Root Properties*.

Notes

1. When Root members are exposed, the first reference or assignment to a member, associates that name (with a nameclass of `~2.6` or `~3.6`) with that member. If, having referenced a member in this way, you subsequently hide Root members using `(2401±0)`, that name remains connected to that member and the member remains exposed. This association may however be removed by erasing the name.
2. If Root members are not exposed, you are free to define an APL object with the same name as one of the members. If you subsequently expose Root members using `(2401±1)`, the name remains associated with the APL object and not with a member of Root. If you then erase the name and re-reference or re-assign it, the name will be associated with the corresponding member.

2.54 Scroll Object

Purpose: Provides a vertical or horizontal scrollbar.

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm ToolBar ToolControl

Children Circle Cursor Ellipse Font Marker Poly Rect Text Timer

Properties Type Posn Size Coord Align Border Active Visible Event Thumb Range Step VScroll HScroll Limits Sizeable Dragable BCol CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex PageSize MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap Select ThumbDrag Scroll

Description

The Scroll object provides a vertical or horizontal scrollbar that can be used as a "free-standing" object or can be "attached" to the side of its parent.

An "attached" scrollbar is one that extends along one edge of a *Form*, *SubForm* or *Group* and has a standard width or height. When the *Form* or *Group* is resized, a vertical attached scrollbar is resized vertically but remains the same width and stays fixed to the side of its parent. Similarly, a horizontal attached scrollbar is resized horizontally but remains the same height.

For most purposes, the use of the Scroll object to provide attached scrollbars in a *Form* has been superseded by the provision of scrollbars as a **property** of a *Form*.

A "free-standing" scrollbar is typically used as a "scale" for selecting a numeric value from a range and may appear and behave rather differently from a standard attached scrollbar. Firstly, a free-standing scrollbar will normally be positioned at an arbitrary position within its parent *Form* or *Group* and be associated with other objects such as *Labels* and *Edit* fields. Secondly, when its parent *Form* or *Group* is resized, it is probably desirable that the scrollbar reacts in the same way as the other child objects, so that the overall appearance of the layout is maintained.

The *Align* property determines whether or not a scrollbar is attached, and if so, to which side of the parent *Group* or *Form* it is fixed. The direction of the scrollbar is determined by the *VScroll* and *HScroll* properties, which are mutually exclusive. The position and size of the scrollbar are determined by *Posn* and *Size*.

To obtain an "attached" scrollbar, it is sufficient for most purposes to specify only the *Align* property. If so, the direction of the scrollbar and its position and size (which are otherwise defined by *VScroll*, *HScroll*, *Posn* and *Size*) are determined automatically for you.

To obtain a "free-standing" scrollbar, it is recommended for most purposes that you set *Align* to 'None' and define the orientation, position and size of the scrollbar explicitly using *VScroll* or *HScroll*, *Posn* and *Size*.

VScroll and *HScroll* may only be set when the object is created and may not subsequently be changed.

If you do attach a "free-standing" scrollbar to a particular side of its parent using *Align*, it will maintain its physical position (in pixels) relative to the side to which it is attached, and its dimension in that direction will remain fixed.

The *Align* property is a character vector containing 'Top', 'Bottom', 'Right', 'Left' or 'None'. If you specify *Align* 'Right' you get a vertical scrollbar attached to the right-hand edge of the parent *Form* or *Group*. *Align* 'Left' also produces a vertical scrollbar, but one that is attached to the left-hand edge. *Align* 'Top' and 'Bottom' each produce horizontal scrollbars, attached to the top and bottom edges of the *Form* or *Group* respectively.

Note that the default value of *Align* is 'Right' unless *HScroll* is set to `-1` in which case it is 'Bottom'. It must therefore be explicitly set to 'None' if you want a non-attached "free-standing" scrollbar.

VScroll and *HScroll* are used to specify the orientation of the scrollbar explicitly, usually in conjunction with *Align* set to 'None'. *VScroll* or *HScroll* may be specified when the object is created by *Dyalog APL Language: Wc*, but cannot be changed using *Dyalog APL Language: Ws*. The two properties are mutually exclusive. Each of them may be set to 0 or `-1`, where `-1` means "true" and 0 means "false". Thus (*VScroll* `-1`) defines a vertical scrollbar, while (*HScroll* `-1`) specifies a horizontal one. Setting either property to `-1` automatically causes the other to be set to 0. If you try to set both to `-1`, *VScroll* takes precedence and *HScroll* is reset to 0.

[Note: the reason for using two properties where one would be sufficient is to allow for the possible future implementation of scrolling groups as provided by `□SM/□SR`.]

Scrolling is controlled by the *Thumb*, *Range* and *Step* properties.

Thumb sets and reports the current position of the "thumb" as an integer in the range 1 to the value of the *Range* property.

Step determines the size of changes reported when the user clicks a scroll arrow (small change) or clicks on the body of the scrollbar (large change). *Step* is a 2-element numeric vector whose first element specifies the value of the "small change" and whose second element specifies the value of the "large change". The *PageSize* property specifies the sizes of the thumb in the scrollbar.

Examples of Attached Scrollbars

```
'F' □WC 'Form' 'Default Scroll Bar'
'F.SCR' □WC 'Scroll'

'F' □WC 'Form' 'Default Horizontal Scroll Bar'
'F.SCR' □WC 'Scroll' ('HScroll' -1)
```

Examples of Free-Standing Scrollbars

```
'F' □WC 'Form' 'Non-Default Scroll Bar'
'F.SCR' □WC 'Scroll' (5 45)(90 10)
    ('Align' 'None')('VScroll' -1)

'F' □WC 'Form' 'Horizontal Scroll Bars'
'F.SC1' □WC 'Scroll' (15 15)(15 70)
    ('Align' 'None')('HScroll' -1)
'F.SC2' □WC 'Scroll' (40 40)(20 20)
    ('Align' 'None')('HScroll' -1)
'F.SC3' □WC 'Scroll' (85 5)(10 90)
    ('Align' 'None')('HScroll' -1)
```

2.55 Separator Object

Purpose: A horizontal or vertical line used to separate items in a menu.

Parents Menu MenuBar

Children Timer

Properties Type Posn Style Event FCol BCol Data EdgeStyle Translate KeepOnClose
MethodList ChildList EventList PropList

Methods Detach

Events Close Create

Description

This object provides a vertical or horizontal line to separate items in a *Menu*. It may also be used to split a *MenuBar* over more than one line.

The orientation of the Separator is determined by its *Style* property, which may be 'Horz' (horizontal) or 'Vert' (vertical). The default is 'Horz'.

If you want to provide a menu with a 3-Dimensional (pushbutton) appearance, you should also set the *EdgeStyle* property on any Separator objects in it. Alternatively, you can achieve the same effect by setting the background colour (*BCol*) for the Separators to grey (192 192 192).

The *Posn* property is a single integer number which specifies the positional index of the Separator relative to the other objects in the *Menu*. A Separator does not generate any events.

Like other components of a menu, the position of a Separator is normally determined by the order in which it is created in relation to other objects with the same parent. However, you can use the *Posn* property to **insert** a Separator into an existing structure. For example, having defined three *MenuItem* objects as children of a *Menu*, you can insert a Separator between the first and the second by specifying its *Posn* to be 2. Note that the value of *Posn* for the *MenuItems* that were previously second and third will then be reset to third and fourth respectively.

If you put a Separator (either *Style*) into a *MenuBar*, it has the effect of adding another line to it. Any items added after the Separator will appear in the new line.

2.56 SM Object

Purpose: Defines a window for □SM / □SR .

Parents Form Group PropertyPage SubForm ToolBar ToolControl

Children Cursor Timer

Properties Type Posn Size Coord Border Visible Event Sizeable Draggable BCol Picture CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj AcceptFiles KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create MouseDown MouseUp MouseDbClick MouseEnter MouseLeave Configure Help DragDrop

Description

This object defines a window for □SM/□SR and allows you to combine the functionality of □SM/□SR with the "windows" GUI. For example, you can define a *Form* with a

MenuBar at the top and a `□SM` window beneath it, with perhaps some *Buttons* alongside.

To allow the user to interact with both `□SM` and other top-level objects, you must specify the names of these objects in the right argument of `□SR`. Thus the statement:

```
CTX ← KEYS CTX □SR 1 2 3 'Form1'
```

allows the user to interact with fields (rows) 1-3 of `□SM` **and** with the object 'Form1' and its children. Callback functions associated with events in 'Form1' will be executed automatically by `□SR`. If an enabled event without a callback occurs, the event will be placed on *Dyalog APL Language: Dq's* internal queue and `□SR` will terminate. The nature of the termination (that is, that it was caused by an event in an object) is reported by the value 131072 (2×17) in the fourth element of `□SR`'s result. The specific event (*Configure*, *MouseUp*, etc.) is however not reported. It is therefore generally preferable to use callbacks.

The *Posn*, *Size* and *Coord* properties allow you to specify the position and size of the window occupied by `□SM` within its parent *Form*. Note however that the `□SM` window will automatically be sized to be an exact number of characters in height and width which will be reported in `□SD`.

The *Border* property may be used to specify a border around the outside of the `□SM` window. It is a number with the value 0 (no border) or 1 (1 pixel border). The default is 0. The *EdgeStyle* property may be used to give the object a 3-dimensional appearance. Its default value is 'Recess'. The area within the `SM` object that is defined by `□SM` is necessarily a multiple of the character size. The region between this area and the outer edges of the object is coloured by the background colour specified by *BCol*, or may be filled with a bitmap specified by *Picture*.

If the user resizes the *Form* which contains the `SM` object, the `SM` object will generate a *Configure* event if enabled. If the *Configure* event is not enabled, `□SR` will terminate with a `RESIZE` error which can be trapped using `□TRAP`. Either method can be used to reformat `□SM` as appropriate.

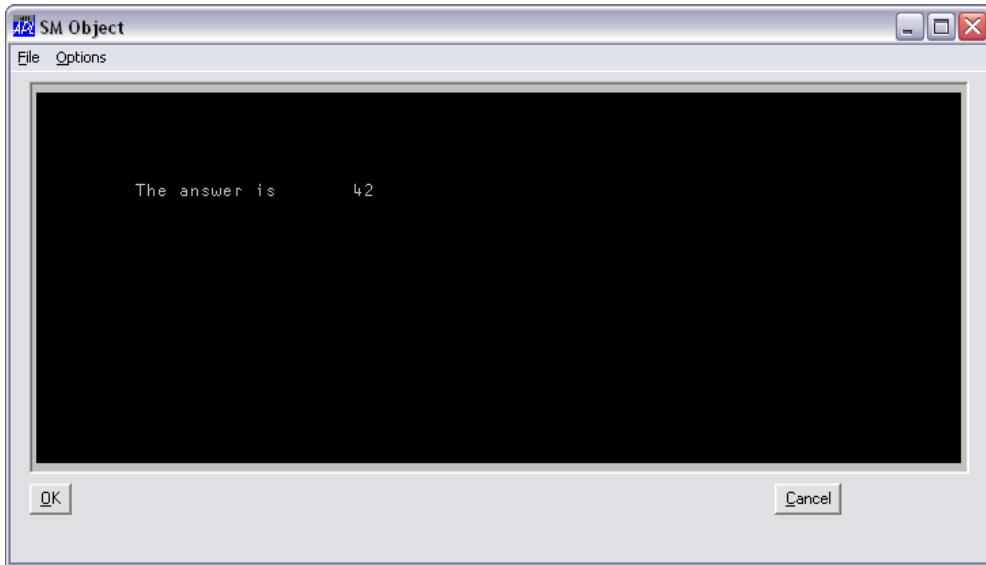
The *MouseDown* event can be used to bring up a pop-up menu. Note however that mouse events are not reported over `□SM` fields because `□SR` uses these to position the cursor.

The illustration shown below was produced as follows :

```

'TEST'      □WC 'Form' 'SM Object' (60 10)(40 50)
'TEST.MB'   □WC 'MenuBar'
'TEST.MB.F' □WC 'Menu' '&File'
'TEST.MB.O' □WC 'Menu' '&Options'
'TEST.B1'   □WC 'Button' '&OK' (84 2)
'TEST.B2'   □WC 'Button' '&Cancel' (84 78)
'TEST.S'     □WC 'SM'(2 2)(80 96)('BCol'192 192 192)
□SM←↑('The answer is' 5 10)(42 5 30)

```



2.57 Spinner Object

Purpose: The Spinner object allows the user to enter a value, using an *UpDown* object to adjust it as required.

Parents ActiveXControl CoolBand Form Grid Group PropertyPage SubForm ToolBar ToolControl

Children Circle Ellipse Font Marker Poly Rect Text Timer

Properties Type Items Text Posn Size Coord Align Border Justify Active Visible Event Thumb Step Wrap Limits Sizeable Dragable FontObj FCol BCol CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj FieldType MaxLength Decimals Password ValidIfEmpty ReadOnly FormatString Changed Value Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap Spin SetSpinnerText Select BadValue KeyError Change

Description

The Spinner object is a special Dyalog APL composite object that consists of an edit field and a pair of spin buttons. The user may enter a value by typing in the edit field and may adjust the value with the spin buttons. The Spinner may cycle through a pre-defined set of values specified by the *Items* property or through a range of values specified by the *Limits* property. The *FieldType* property supports all of the standard data types, that is, Char, Numeric, LongNumeric, Currency, Date, LongDate, and Time.

The *Limits* property is a 2-element numeric vector that specifies the minimum and maximum value of the object. The *Step* property specifies the amount by which the value is incremented or decremented by the spin buttons. The current value in the object is defined by the *Thumb* and *Value* properties, which are usually identical. If *ReadOnly* is 0, the user may type a value into the edit field which will be validated and converted according to the *FieldType*. In this case, the *Value* and the *Thumb* properties may be different.

An *alternative* way to use the Spinner object is to specify the *Items* property. This defines a discrete set of values through which the user may cycle, and the object behaves rather like a *Combo* without a drop-down list. In this case, the *Limits* property is automatically set to $(1, \rho \text{Items})$, *Thumb* refers to the *index* into the list of *Items*, and *Step* specifies the amount by which this index is updated by the spin buttons. For example, if you set *Step* to 3, the spin buttons would display every third item. The *Items* property may be a character matrix, a vector of character vectors, or a numeric vector and will be formatted according to the *FieldType*. For example, if you wanted the user to select one of a set of specific dates, you would set the *FieldType* to Date or LongDate and the *Items* property to the day numbers (since 1 January 1900) corresponding to the dates you require. The *ReadOnly* property specifies whether or not the user may enter data into the edit field. A value typed in by the user will be converted and formatted according to the *FieldType* but need not correspond to a value in *Items*.

In operation, the value in the Spinner is adjusted by the *Step* each time one of the spin buttons is clicked. If the user holds a spin button down, the value is adjusted at the rate defined for the keyboard repeat rate. Furthermore, the size of each adjustment is increased according to the length of time the button stays depressed. After 1 second, the amount is increased to $(2 \times \text{Step})$. after 2 seconds, to $(4 \times \text{Step})$, after 3

seconds to $(8 \times \text{Step})$ and so forth until the amount of adjustment exceeds one quarter of the range ($\text{Limits}[2] \div \text{Limits}[1]$).

When the value in the spinner reaches its top or bottom limit, it will wrap around to the opposite limit if the value of the *Wrap* property is 1 (the default). Otherwise it will stick.

The *MaxLength* property defines the maximum number of characters that the user may type into the edit field. The *Decimals* property specifies the number of decimal places to which a numeric value is displayed and applies only if the *FieldType* is Numeric or LongNumeric.

The Spinner generates two special events, *Spin* and *SetSpinnerText*. The *Spin* event is generated each time the value of the *Thumb* is about to be updated and reports the new value and the difference between it and the current value. You may prevent the *Thumb* from being updated by returning 0 from a callback function, or you may alter the new value of the *Thumb* by returning a modified message. The *SetSpinnerText* event is generated after the *Thumb* has been reset but *before* the edit field has been updated. It reports the new value of the *Thumb* and the text that is about to be written into the edit field. By returning a modified event message from a callback, this event allows your application to respond *dynamically* to the spin buttons and to control the text in the edit field directly.

Like an *Edit* object, the Spinner has a *Changed* property and generates a Change event when loses the focus after the value of its *Text* and/or *Thumb* property has been altered.

If *FieldType* is Numeric, LongNumeric, Currency, Date, LongDate or Time, the Spinner will generate a *BadValue* event when it loses the focus if the text in the edit field (that is, the *Text* property) is in conflict with the *FieldType* property and cannot be converted to an appropriate number, or is outside the range specified by the *Limits* property. If the edit field is empty, a *BadValue* event will be generated if *ValidIfEmpty* is 0, but not if it is set to 1.

2.58 Splitter Object

Purpose: The Splitter object divides a container into resizable panes.

Parents ActiveXControl Form Group PropertyPage SubForm

Children Timer

Properties Type SplitObj1 SplitObj2 Posn Size Style Coord Align Active Visible Event
BCol CursorObj Data KeepOnClose MethodList ChildList EventList PropList

Methods Detach

Events Close Create StartSplit Splitting EndSplit

Description

The Splitter divides the client area of a *Form* or *SubForm* into resizable panes. Each pane created this way may be empty or be occupied by a single object. If the object in a pane is itself a container object, such as a *SubForm*, it may have a number of other controls within it.

A single Splitter may manage the geometry of 0, 1 or 2 other objects, which, together with the Splitter itself, share the same parent. The two objects are named by the *SplitObj1* and *SplitObj2* properties respectively.

A Splitter may manage objects of the following types:

Button	Calendar	Combo	Edit	Grid	Group
Label	List	ListView	MDIClient	ProgressBar	RichEdit
Scroll	Spinner	Static	StatusBar	SubForm	TabBar
TabControl	ToolBar	TrackBar	TreeView	UpDown	

If *Style* is 'Vert' (the default), the Splitter is drawn vertically in its parent with the first object (*SplitObj1*) positioned to its left, and the second object (*SplitObj2*) to its right. See *Splitter: Example 1*.

If *Style* is 'Horz', the Splitter is drawn horizontally in its parent with the first object (*SplitObj1*) positioned above, and the second object (*SplitObj2*) below. See *Splitter: Example 2*.

The *Style* property must be set when the object is created with `□WC` and may not subsequently be changed using `□WS`.

The *Posn* and *Size* properties are partially read-only, in that only one dimension of the value may be specified. If *Style* is 'Vert', you may specify the x-coordinate and the width of the Splitter, but you may not specify its y-coordinate nor its height. If *Style* is 'Horz', you may specify the y-coordinate and the width of the Splitter, but you may not specify its x-coordinate nor its length.

When the user positions the mouse pointer directly over the Splitter object, the cursor changes (by default) to a double-headed arrow (direction in accordance with *Style*). The user may now depress the left mouse button and drag the Splitter to a new position, resizing the objects named by *SplitObj1* and *SplitObj2* in the process.

You can select a different cursor using the *CursorObj* property. Note that setting the *CursorObj* property to 0 selects the default cursor, which is the appropriate double-headed arrow.

When the user depresses the mouse button, the Splitter generates a *StartSplit* event. When the user releases the mouse button, the Splitter generates an *EndSplit* event. If full-drag is in effect, the Splitter also reports *Splitting* events as it is dragged. All these events report the new or current position of the Splitter object and are provided for information only.

Note that the objects named by *SplitObj1* and *SplitObj2* and any sub-objects they contain will generate Configure events when they are resized by the Splitter.

Alignment

The *Align* property specifies how a Splitter behaves when its parent is resized and may be 'None', 'Left', 'Right', 'Top' or 'Bottom'.

If *Align* is 'None', the Splitter moves as its parent is resized, so that it divides its parent in the same *proportions* as before. This is the default.

Any other value of *Align* attaches the Splitter to the corresponding edge of its parent. For example, if *Align* is 'Left', the width of the object to the *left* of the Splitter remains fixed when its parent is resized horizontally by the user.

Like the *Style* property, *Align* may be set only when the object is created with `□WC` and may not subsequently be changed using `□WS`.

Using Multiple Splitters

If you want to divide a Form into more than 2 resizable panes, there are two possible approaches, each with its own different characteristics.

The first approach is a hierarchical one using SubForms. This example shows how you can create a Form containing three resizable Edit objects.

First, you create an Edit, a SubForm, and a Splitter as children of the *Form*, using the Splitter to divide the Form into two panes, one for the Edit and the other for the

SubForm. Next, you create two Edit objects and a Splitter as children of the *SubForm*, using the second Splitter to divide the SubForm into two. You can continue with this approach to any reasonable depth. See *Splitter: Example 3*.

Notice that, by default, when the first Splitter is shifted to the left, both panes in the SubForm expand equally.

The second approach is to create multiple Splitters *at the same level*, that is, owned by the same parent. See *Splitter: Example 4*.

In this case, the third Edit object *F . E3* is unaffected by movement of the leftmost Splitter *F . S1*. Note also, that the first Splitter *F . S1* may not be dragged further right than the second Splitter *F . S2*.

Using the non-hierarchical approach, horizontal and vertical Splitters may be combined in interesting ways. This can also be achieved using nested SubForms, but at the expense of a complex object hierarchy. See *Splitter: Example 5*.

Notice that in this example, with the exception of the last Splitter *F . S4*, it is necessary only to specify the *SplitObj1* property for each of the Splitters. The reason is that the first four Splitters only manage one object *directly*. For example, the object to the right of *F . S1* is in fact a horizontal Splitter *F . S2*. Dragging *F . S1* changes the length of *F . S2* which in turn changes the width of *F . E2*. and *F . E3*.

Colliding Splitters

If you have two or more vertical Splitters or two or more horizontal Splitters in the same parent object, it is possible for the user to make the Splitters *collide*. This can occur by dragging one of the Splitters into the other, or, unless both Splitters have *Align* set to 'None', by shrinking the parent.

When Splitters collide, the object being dragged by the user (a Splitter or a border of the parent) takes precedence over the setting of *Align*, and temporarily *pushes* other Splitters along in its direction of travel. If and when the operation is reversed, the other Splitters are *pulled* back to their original positions.

2.59 Static Object

Purpose: This object is primarily used to display graphics in a sub-window.

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm ToolBar
ToolControl

Children Circle Cursor Ellipse Font Image Locator Marker Metafile Poly Rect Text Timer

Properties Type Posn Size Style Coord Border Active Visible Event Sizeable Dragable
FontObj FCol BCol Picture CursorObj AutoConf YRange XRange Data Attach TextSize
EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles
KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ChooseFont

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose
Help MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave
MouseWheel FontOK FontCancel Select

Description

The overall appearance of an empty Static object is controlled by the value of its *Style* property which may be one of the following character vectors:

'BlackFrame'	'BlackBox'
'GreyFrame' or 'GrayFrame'	'GreyBox' or 'GrayBox'
'WhiteFrame'	'WhiteBox'

Note that the colours implied by the *Style* are not "hard-coded" but are actually defined by the current Windows colour scheme as follows:

Black	Window Border Colour
Grey/Gray	Desktop Colour
White	Window Background Colour

If the background colour of the *Form* is also set to the Window Background Colour, it follows that the *Styles* 'WhiteFrame' and 'WhiteBox' make the Static itself invisible (against the background), although the **contents** of the Static will show. This makes the Static appear like an invisible clipping window.

2.60 StatusBar Object

Purpose: This object is used to manage *StatusField* objects which display information for the user.

Parents ActiveXControl CoolBand Form SubForm

Children Bitmap BrowseBox Circle Cursor Ellipse FileBox Font Icon Image Marker Poly ProgressBar Rect StatusField Text Timer

Properties Type Posn Size Coord Align Border Active Visible Event VScroll HScroll Sizeable FontObj FCol BCol Picture CursorObj AutoConf YRange XRange Data Attach TextSize EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

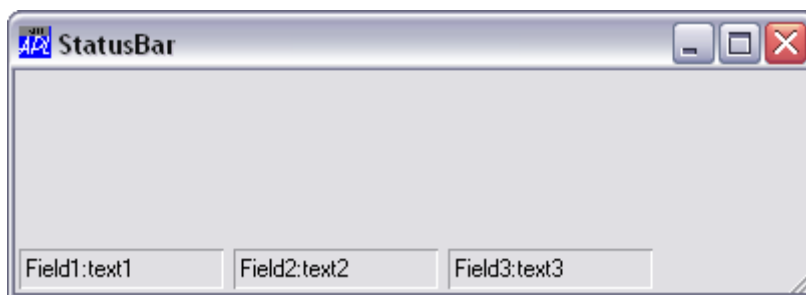
Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ChooseFont

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel Select

Description

The *StatusBar* is a container object that manages *StatusFields*. *StatusField* objects display textual information and are typically used for help messages and for monitoring the status of an application. They can also be used to automatically report the status of the Caps Lock, Num Lock, Scroll Lock, and Insert keys.

```
'TEST'□WC'Form' 'StatusBar' ('EdgeStyle' 'Default')
'TEST.SB'□WC'StatusBar'
'TEST.SB.S1'□WC'StatusField' 'Field1:' 'text1'
'TEST.SB.S2'□WC'StatusField' 'Field2:' 'text2'
'TEST.SB.S3'□WC'StatusField' 'Field3:' 'text3'
```



The *Align* property determines to which side of the parent *Form* or *SubForm* the *StatusBar* is attached. By default, a *StatusBar* is positioned along the lower edge of the *Form* (*Align* 'Bottom'). Using the *Align*, *Posn* and *Size* properties you may create *StatusBars* in different positions and with differing sizes if you wish. Notice that the *Align* property controls how the *StatusBar* reacts to its parent *Form* being resized. If *Align* is 'Top' or 'Bottom', the *StatusBar* remains fixed in height but stretches and shrinks sideways with the *Form*. If *Align* is 'Left' or 'Right', the *StatusBar* remains fixed in width and stretches and shrinks vertically with the *Form*.

By default a *StatusBar* has a Button Face colour background and the value of its *EdgeStyle* property is 'Default'. This gives it the appearance shown above.

Unless you specify the position and size of its children, a *StatusBar* automatically manages their geometry. The first *StatusField* is positioned just inside its top left corner. If *Align* is 'Top' or 'Bottom', the next *StatusField* is positioned alongside the first but with a small gap between them. Subsequent *StatusFields* are added in a similar fashion. If *Align* is 'Left' or 'Right', the second and subsequent *StatusFields* are added below the first with a similar gap between them. In either case you can position and size the *StatusFields* explicitly if you wish.

If you attempt to add a *StatusField* that would extend beyond the right edge (*Align* 'Top' or 'Bottom') or bottom edge (*Align* 'Left' or 'Right') the behaviour depends upon the value of *HScroll* or *VScroll*. If *HScroll* is 0 (the default) and *Align* is 'Top' or 'Bottom', the *StatusField* is added below the first one, thereby starting a new row. If *VScroll* is 0 (the default) and *Align* is 'Left' or 'Right', it is added to the right of the first one thereby starting a new column. If *HScroll* or *VScroll* is -1 or -2, the new *StatusField* is simply positioned in the same row or column and may be scrolled into view using a mini scrollbar. A value for *HScroll* or *VScroll* of -1 causes the mini scrollbar to be permanently present in the *Scroll Bar*. A value of -2 causes it to appear only when required.

VScroll and *HScroll* may only be set when the object is created and may not subsequently be changed.

2.61 StatusField Object

Purpose: This object is used to display information for the user.

Parents StatusBar

Children Menu Timer

Properties Type Caption Text Posn Size Style Coord Border Visible Event Sizeable Draggable FontObj FCol BCol Picture AutoConf Data Attach EdgeStyle Translate Accelerator KeepOnClose MethodList ChildList EventList PropList

Methods Detach

Events Close Create DropObjects Select MouseDown MouseUp MouseMove MouseDblClick

Description

The StatusField object provides an area for displaying context sensitive help messages, keyboard status, and other application dependent information.

By default a StatusField is a recessed rectangle in which information is displayed. It has a *Caption* and a *Text* property, which by default are empty, but either or both of which can be used to present information. The *Caption* is left justified in the field and the *Text* is displayed immediately to its right. Typically, you would use the *Caption* property as a title to describe the information that the StatusField displays, and the *Text* property to show its current value. However, you are not obliged to use both of them and you can achieve most effects with just one.

Note that when the StatusField is used to display hints it is its *Text* property that is used.

A StatusField may be used to monitor the status of the keyboard and this is controlled by its *Style* property. The default value for *Style* is an empty vector. However, you can set it to monitor various keyboard states as follows :

CapsLock	Monitors state of Caps Lock key
ScrollLock	Monitors state of Scroll Lock key
NumLock	Monitors state of Num Lock key
KeyMode	Monitors the keyboard mode (APL/ASCII)
InsRep	Monitors the state of the Insert key

In each case, the *Text* property of the StatusField is used to display the keyboard status. If *Style* is CapsLock, ScrollLock or NumLock, the field displays "Caps", "Num" or "Scroll" if the corresponding mode is selected and is blank if not.

If *Style* is InsRep, the StatusField displays either "Ins" or "Rep". Initially it always displays "Ins" and then toggles between "Rep" and "Ins" each time the Insert key is pressed.

If *Style* is KeyMode, the StatusField displays the name for the current keyboard mode which is defined in the input table being used. For the 2-mode tables APL_US.DIN, APL_UK.DIN etc., the mode name displayed is either "Apl" or "Asc". The unified tables have no modes so a StatusField with this *Style* does nothing.

If *Style* is set to one of the above, you may still use the *Caption* property to give the StatusField a title. You may even set the value of the *Text* property, but be aware that this value will be reset when the user next presses the key the StatusField is monitoring.

2.62 SubForm Object

Purpose: This object represents a window that is owned by and constrained within another *Form* or an *MDIClient* .

Parents ActiveXControl CoolBand Form Group MDIClient PropertyPage SubForm TabControl ToolBar ToolControl

Children Animation Bitmap BrowseBox Button ButtonEdit Calendar Circle ColorButton Combo ComboEx Cursor DateTimePicker Edit Ellipse FileBox Font Form Grid Group HTMLRenderer Icon Image ImageList Label List ListView Locator Marker MDIClient Menu MenuBar Metafile MsgBox OCXClass Poly ProgressBar PropertySheet Rect RichEdit Scroll SM Spinner Splitter Static StatusBar SubForm TabBar TabControl TCPSocket Text Timer TipField ToolBar ToolControl TrackBar TreeView UpDown

Properties Type Caption Posn Size Coord State Border Active Visible Event Thumb Range Step VScroll HScroll Sizeable Moveable SysMenu MaxButton MinButton HelpButton FontObj BCol Picture OnTop IconObj CursorObj AutoConf YRange XRange Data Attach TextSize EdgeStyle Handle Hint HintObj Tip TipObj TabObj Translate Accelerator AcceptFiles KeepOnClose Dockable Docked DockShowCaption DockChildren UndocksToRoot Redraw TabIndex PageSize MethodList ChildList EventList PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDblClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap StateChange MDIActivate MDIDeactivate DockStart DockMove DockRequest DockAccept DockEnd DockCancel Select FrameContextMenu VScroll HScroll VThumbDrag HThumbDrag

Description

If the SubForm is the child of a *Form*, it is by default a simple featureless window that occupies the entire client area (excluding standard *ToolBars*, *StatusBars* and *TabBars*) of its parent.

The properties that control its appearance, including *Sizeable*, *Moveable*, *SysMenu*, *Border*, *MaxButton* and *MinButton*, all default to 0. The *EdgeStyle* property also defaults to 'None', so the background of the SubForm defaults to the Window Background colour.

If the SubForm is the child of an *MDIClient*, its default appearance is the same as for a top-level *Form*. By default its size is 25% of its parent client area and it is positioned in the centre of its parent object.

The *Posn* property specifies the location of the **internal** top-left corner of the SubForm relative to its parent. If the SubForm has a title bar, border, or a 3-dimensional shadow, you must allow sufficient space for these components. Similarly, the *Size* property specifies the internal size of the SubForm excluding the title bar and border.

A SubForm is constrained so that it cannot be moved outside its parent. In all other respects it behaves in a similar manner to a *Form* object. See *Form* object and the descriptions of its properties for further details.

2.63 SysTrayItem Object

Purpose: The SysTrayItem object represents an item that you can create in the Windows System Tray.

Parents Form Root

Children Icon Menu Timer

Properties Type Event IconObj Data Tip Translate Popup KeepOnClose MethodList ChildList EventList PropList

Methods Detach ShowBalloonTip Wait

Events Close Create MouseDown MouseUp MouseMove MouseDbClick BalloonShow BalloonHide BalloonTimeout BalloonUserClick

Description

The SysTrayItem object appears as an icon in the Windows System Tray and allows the user to interact with your application even if it is minimised or has no other visible presence.

Interaction is provided through a pop-up menu that is displayed when the user clicks on the SysTrayItem. The SysTrayItem does not support mouse or keyboard events directly.

The IconObj property specifies the name of an Icon object used to display the SysTrayItem. If not specified, the default is the standard Dyalog APL icon.

The `Popup` property specifies the name of a `Menu` object (which may be a child of the `SysTrayItem`). The `Menu` object is displayed automatically when the user clicks on the `SysTrayItem` icon. The `Menu` should contain one or more `MenuItem` objects with suitable callback functions attached.

Unlike other popup menus, the `SysTrayItem` menu is not activated by an explicit (modal) `□□□` but is posted automatically for you. The `MenuItem` callbacks will be executed by the current `□□□`, with the exception of modal `□□□`s on `MsgBox`, `FileBox`, `Locator` and other popup `Menu` objects. For example, if your application is in a modal `□□□` on a `Form`, that `□□□` will react to and action events on the `SysTrayItem` menu, even though it is not explicitly included in the list of objects being `□□□`'ed.

The `Tip` property specifies a character string to be displayed when the user hovers the mouse over the `SysTrayItem`. This is displayed using the user's current setting for `Tip` text and it is not possible to change this appearance.

2.64 TabBar Object

Purpose: To manage a set of *TabBtn* objects.

Parents `ActiveXControl` `Form` `SubForm`

Children `Circle` `Ellipse` `Font` `Marker` `Poly` `Rect` `TabBtn` `Text` `Timer`

Properties `Type` `Posn` `Size` `Coord` `Align` `Active` `Visible` `Event` `VScroll` `HScroll` `Sizeable` `FontObj` `BCol` `Picture` `OnTop` `IconObj` `CursorObj` `AutoConf` `YRange` `XRange` `Data` `Attach` `TextSize` `Handle` `Hint` `HintObj` `Tip` `TipObj` `TabObj` `Translate` `Accelerator` `AcceptFiles` `KeepOnClose` `Redraw` `TabIndex` `MethodList` `ChildList` `EventList` `PropList`

Methods `Detach` `GetTextSize` `Animate` `GetFocus` `ShowSIP` `GetFocusObj`

Events `Close` `Create` `DragDrop` `Configure` `ContextMenu` `DropFiles` `DropObjects` `Expose` `Help` `MouseDown` `MouseUp` `MouseMove` `MouseButtonClick` `MouseEnter` `MouseLeave` `MouseWheel` `Select`

Description

The `TabBar` object manages a group of *TabBtn* objects. These are associated with a set of *SubForm* objects which are positioned on top of one another. When the user clicks on a *TabBtn*, the corresponding *SubForm* is brought to the top and given the focus.

TabBar and *TabBtn* objects were implemented before Windows provided direct support for tabbed dialogs, and have been superseded by *TabControl* and *TabButton* objects. Please use these instead.

By default, a *TabBar* is a flat bar stretched across the bottom of its parent form. You can alter its appearance using its *EdgeStyle* property and you can control its alignment with its *Align* property. *Align* can be set to *Top*, *Bottom* (the default), *Left* or *Right* and causes the *TabBar* to be attached to the corresponding edge of the *Form*. A *TabBar* aligned *Top* or *Bottom* will automatically stretch or shrink horizontally when its parent *Form* is resized, but it will remain fixed vertically. A *TabBar* aligned *Left* or *Right* will stretch vertically but will remain fixed horizontally. By default a *TabBar* occupies the entire width or length of the side of the *Form* to which it is attached. Both the *Posn* and *Size* properties can be altered.

The alignment of a *TabBar* also determines the orientation of its *TabBtns*. *TabBars* aligned *Top* or *Bottom* cause their *TabBtns* to be drawn left to right with the free edge of the *TabBtns* facing downwards or upwards respectively. *TabBar* aligned *Left* or *Right* draw their *TabBtns* downwards with their free edges facing left or right respectively.

By default, *TabBtn* objects are positioned along the inner edge of the *TabBar*. This is the edge closest to the *SubForm*s they will tab. They are also positioned so that they overlap one another horizontally or vertically according to the *Align* property.

The *HScroll* and *VScroll* properties determine what happens when the end of the *TabBar* is reached. If *HScroll* or *VScroll* is 0 (the default) a *TabBtn* that would otherwise extend beyond the *TabBar* is instead positioned immediately above, below or alongside the first *TabBtn* in the *TabBar*, thereby starting a new row or column. Note however that the *TabBar* is not automatically resized vertically to accommodate a second row or column. If you want a multi-flight *TabBar* you have to set its height or width explicitly. If *HScroll* or *VScroll* is -1 or -2, *TabBtns* continue to be added along the *TabBar* even though they extend beyond its boundary and may be scrolled into view using a mini scrollbar. If *HScroll* is -1, the scrollbar is shown whether or not any controls extend beyond the *TabBar*. If *HScroll* is -2, the scrollbar appears only if required and may appear or disappear when the user resizes the parent *Form*.

VScroll and *HScroll* may only be set when the object is created and may not subsequently be changed.

If you specify a value for its *Posn* property, a *TabBtn* will be placed at the requested position regardless of the value of *Style*, *HScroll* or *VScroll*. However, the next control added will take its default position from the previous one according to the value of these properties. Thus if you wish to group your controls together with spaces between the groups, you need only specify the position of the first one in each group.

If you specify a value for its *Posn* property, a *TabBtn* will be placed at the requested position regardless of the value of *Align*. However, the next *TabBtn* added will take its default position from the previous one. Thus if you wish to group your *TabBtns* together with spaces between the groups, you need only specify the position of the first one in each group.

2.65 TabBtn Object

Purpose: To tab a *SubForm* .

Parents *TabBar*

Children *Timer*

Properties *Type* *Caption* *Posn* *Size* *Align* *Border* *Active* *Visible* *Event* *FontObj* *FCol* *BCol* *AutoConf* *Data* *Attach* *EdgeStyle* *TabObj* *Translate* *Accelerator* *KeepOnClose* *MethodList* *ChildList* *EventList* *PropList*

Methods *Detach* *ChooseFont*

Events *Close* *Create* *FontOK* *FontCancel* *DropObjects* *Select* *MouseDown* *MouseUp* *MouseMove* *MouseDownClick*

Description

TabBtn objects are associated with *SubForms* which are positioned on top of one another. When the user clicks on a *TabBtn*, the corresponding *SubForm* is brought to the top and given the focus.

TabBar and *TabBtn* objects were implemented before Windows provided direct support for tabbed dialogs, and have been superseded by *TabControl* and *TabButton* objects. Please use these instead.

The appearance of a *TabBtn* is determined by its *EdgeStyle*, *Border* and *Caption* properties. These take their defaults from the *SubForm* with which the *TabBtn* is associated. Thus there is generally no need to specify them. *BCol* also defaults to that of its associated *SubForm*.

The position of a *TabBtn* is normally determined by its parent *TabBar* and its default size is fixed (22 x 80 pixels), and not related to the size of its *Caption*. These defaults can be overridden using the *Posn* and *Size* properties.

A *SubForm* is associated with a *TabBtn* by setting the *TabObj* property of the *SubForm* to the name of, or ref to, the *TabBtn*. The *TabObj* property of the *TabBtn* is a read-only property that contains the name of, or ref to, the associated *SubForm*.

2.66 TabButton Object

Purpose: The *TabButton* object represents an individual tab or button in a *TabControl*

Parents *TabControl*

Children *Timer*

Properties *Type* *Caption* *Posn* *Size* *State* *Event* *ImageIndex* *Data* *Tip* *TabObj* *Accelerator* *KeepOnClose* *MethodList* *ChildList* *EventList* *PropList*

Methods *Detach*

Events *Close* *Create* *Select*

Description

The *TabButton* object represents an individual tab or button in a *TabControl*

The position and size of a *TabButton* object are entirely determined by its parent *TabControl* and may not be altered. For this reason, the *Posn* and *Size* properties are read-only.

The *Caption* property specifies the text that appears on the button or tab.

A picture is specified by setting the *ImageIndex* property of the *TabButton*. This is a number that points to a particular icon or bitmap defined in an *ImageList* object whose name is specified by the *ImageListObj* property of the parent *TabControl*.

Note that all *TabButton* objects share the same font which is defined by the *FontObj* property of the *TabControl*.

The foreground and background colours of the *TabButton* object are fixed.

When used as a tab, a *TabButton* is normally attached to a *SubForm* by the *TabObj* property of the *SubForm*. The *TabObj* property of the *TabButton* itself, is a read-only property that specifies the name of, or ref to, the *SubForm* to which the *TabButton* is attached.

The *State* property reports the (selected) state of a *TabButton*.

2.67 TabControl Object

Purpose: The TabControl object provides access to the native Windows tab control.

Parents ActiveXControl CoolBand Form SubForm

Children ImageList SubForm TabButton Timer

Properties Type Posn Size Style Visible Event ImageListObj FontObj Data Attach Handle TabObj KeepOnClose MultiLine TabSize Justify TabJustify Align MultiSelect TabFocus HotTrack ScrollOpposite FlatSeparators MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create

Description

The standard tab control is analogous to a set of dividers in a notebook and allows you to define a set of *pages* that occupy the same area of a window or dialog box. 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. For example, the Windows taskbar is such a tab control.

The overall appearance of the TabControl is determined by the *Style* property which may be 'Tabs' (the default), 'Buttons' or 'FlatButtons'.

Individual tabs or buttons are represented by *TabButton* objects which should be created as children of the TabControl object. Optional captions and pictures are specified by the *Caption* and *ImageIndex* properties of the individual *TabButton* objects themselves. Otherwise, the appearance of the tabs or buttons is determined by properties of the TabControl itself.

To implement a multiple page tabbed dialog (*see Example 1*), you should create a *Form*, then a TabControl with Style 'Tabs' as a child of the *Form*. Next, create one or more pairs of *TabButton* and *SubForm* objects as children of the TabControl. You associate each *SubForm* with a particular tab by setting its *TabObj* property to the name of, or ref to, the associated *TabButton* object. Making the *SubForms* children of the TabControl ensures that, by default, they will automatically be resized correctly. You may alternatively create your *SubForms* as children of the main *Form* and establish appropriate resize behaviour using their *Attach* property.

A `TabControl` object with *Style* 'Buttons' (Example 2) or 'FlatButtons' (Example 3) may be used in a similar way (that is, to display a set of alternative pages), although buttons in this type of `TabControl` are more normally used to execute commands. For this reason, these styles of `TabControl` are without borders.

If *Style* is 'FlatButtons', the *FlatSeparators* property specifies whether or not separators are drawn between the buttons. The default value of *FlatSeparators* is 0 (no separators).

Example 3a shows the effect of setting *FlatSeparators* to 1.

The *Align* property specifies along which of the 4 edges of the `TabControl` the tabs or buttons are arranged. *Align* also controls the relative positioning of the picture and Caption within each `TabButton`. *Align* may be Top (the default), Bottom, Left or Right.

If *Align* is 'Top' or 'Bottom' (Example 4), the tabs or buttons are arranged along the top or bottom edge of the `TabControl` and picture is drawn to the left of the Caption.

If *Align* is 'Left' (Example 5), the tabs or buttons are arranged top-to-bottom along the left edge of the `TabControl`, and the pictures are drawn below the Captions.

In recent versions of Windows, 'Align' 'Right' fails to show the tabs or buttons on the right hand edge of the `TabControl`; this appears to be a limitation of Windows.

The *Attach* property specifies how the `TabControl` responds when its parent is resized. Its default value, which is independent of the *Align* property, is 'None' 'None' 'None' 'None'. This causes the `TabControl` to maintain its original proportions when its parent is resized.

The *MultiLine* property determines whether or not your tabs or buttons will be arranged in multiple flights or multiple rows/columns.

The default value of *MultiLine* is 0, in which case, if you have more tabs or buttons than will fit in the space provided, the `TabControl` displays an UpDown control to permit the user to scroll them. See Example 7.

If *MultiLine* is set to 1, the tabs are displayed in multiple flights (Example 8) or the buttons are displayed in multiple rows (Example 9).

The *ScrollOpposite* property specifies that unneeded tabs scroll to the opposite side of a `TabControl`, when a tab is selected. Setting *ScrollOpposite* to 1 forces *MultiLine* to 1 also.

Example 10 illustrates a `TabControl` with `ScrollOpposite` set to 1, after the user has clicked *Third Tab*. Notice that, in this example, the `SubForms` have been created as children of the `TabControl`. This is necessary to ensure that they are managed correctly in this case.

If `MultiLine` is 1, the way that multiple flights of tabs or rows/columns of buttons are displayed is further defined by the `Justify` property which may be `'Right'` (the default) or `'None'`.

If `Justify` is `'Right'` (which is the default), the `TabControl` increases the width of each tab, if necessary, so that each row of tabs fills the entire width of the tab control. Otherwise, if `Justify` is empty or `'None'`, the rows are ragged.

See *Example 11* (for tabs) and *Example 12* (for buttons).

By default, the size of the tabs may vary from one to another. Fixed size tabs may be obtained by setting the `TabSize` property.

To obtain fixed sized tabs with `MultiLine` set to 1, you must however also set `Justify` to `'None'`.

If fixed size tabs are in effect, the positions at which the picture and Caption are drawn within each `TabButton` is controlled by the `TabJustify` property which may be `'Centre'`, `'Edge'`, or `'IconEdge'`.

Example 13 illustrates these different settings.

The font used to draw the captions in the `TabButton` objects is determined by the `FontObj` property of the `TabControl`.

You cannot specify the foreground or background colours of the tabs/buttons, nor can you use different fonts in different tabs/buttons. The orientation of the Caption text is always determined by the value of the `Align` property of the `TabControl`.

The `TabObj` property is read-only and reports the name of, or ref to, the `TabButton` that is currently selected.

The `MultiSelect` property specifies whether or not the user can select more than one button in a `TabControl` at the same time, by holding down the `Ctrl` key when clicking. The default is 0 (only one button may be selected). `MultiSelect` is ignored if `Style` is `'Tabs'`.

The `TabFocus` property specifies the focus behaviour for the `TabControl` object and may be `'Normal'` (the default), `'Never'` or `'ButtonDown'`.

The *HotTrack* property specifies whether or not the tabs or buttons are automatically highlighted by the mouse pointer. The default is 0 (no highlighting).

2.68 TCPSocket Object

Purpose: The TCPSocket object provides an interface to TCP/IP.

Parents ActiveXControl Calendar CoolBand DateTimePicker Form NetType OLEClient OLEServer PropertyPage Root SubForm TCPSocket

Children Bitmap BrowseBox Clipboard Cursor FileBox Font Form Icon ImageList Locator Menu Metafile MsgBox OCXClass OLEClient Printer PropertySheet TCPSocket Timer TipField

Properties Type LocalAddr LocalPort RemoteAddr RemotePort Style Event LocalAddrName LocalPortName RemoteAddrName RemotePortName Data SocketType SocketNumber CurrentState TargetState KeepOnClose Encoding MethodList ChildList EventList PropList

Methods Detach TCPSend TCPGetHostID TCPSendPicture Wait

Events Close Create TCPError TCPAccept TCPConnect TCPRecv TCPClose TCPGotAddr TCPGotPort TCPReady

Description

The TCPSocket object provides an event-driven mechanism to communicate with other programs (including Dyalog APL) via TCP sockets. Dyalog recommends that Conga is used in preference to TCPSockets in new applications.

The *SocketType* property is a character vector that specifies the type of the TCP/IP socket. This is either 'stream' (the default), or 'UDP'. *SocketType* must be defined when the object is created and cannot be set or changed using `⌵WS`.

The *Style* property is a character vector that specifies the type of data transmitted or received by the socket; it may be 'Char', 'Raw', or 'APL'. The value 'APL' is valid only if the *SocketType* is 'stream'.

The *Encoding* property is a character vector that specifies how character data are encoded or translated. The possible values are 'None', 'UTF-8', 'Classic' or 'Unicode', depending upon the value of the *Style* property.

LocalAddr and *LocalPort* properties identify your end of the connection; *RemoteAddr* and *RemotePort* identify the other end of the connection. The values of the two sets of

properties are clearly symmetrical; your *LocalAddr* is your partner's *RemoteAddr*, and there are strict rules concerning which of them you and your partner may set. See the individual descriptions of these properties for details.

The *SocketNumber* property is the Window handle of the socket attached to the *TCP Socket* object and is generally a read-only property. The only time that *SocketNumber* may be specified is when a server replicates (clones) a listening socket to which a client has just connected.

2.69 Text Object

Purpose: Writes text.

Parents ActiveXControl Animation Bitmap Button ButtonEdit Combo ComboEx Edit Form Grid Group Label List ListView MDIClient Metafile Printer ProgressBar PropertyPage PropertySheet RichEdit Scroll Spinner Static StatusBar SubForm TabBar TipField ToolBar TrackBar TreeView UpDown

Children Timer

Properties Type Text Points FCol BCol VAlign HAlign Coord Active Visible Event Draggable FontObj OnTop CursorObj AutoConf Data Translate Accelerator KeepOnClose DrawMode MethodList ChildList EventList PropList

Methods Detach ChooseFont

Events Close Create FontOK FontCancel DragDrop MouseDown MouseUp MouseMove MouseDbClick Help Select

Description

The Text object is used to write arbitrary text. It can be used in a *Form*, *SubForm* or *Group* instead of a *Label*. The main difference is that a *Label* is implemented as a true window object (thus consuming Windows resources). A Text object is not a window and consumes no MS-Windows resources. However, a *Label* supports *DragDrop* events and has various useful properties that are not shared by the Text object.

The contents of the Text object are defined by its *Text* property. This is a character array containing one of the following :

- a simple scalar
- an enclosed vector or matrix (also a scalar)
- a simple vector

- a simple matrix
- a vector of enclosed vectors or matrices

Points is either a simple 2-column matrix of (y,x) co-ordinates, or a 2-element vector of y-coordinates and x-coordinates respectively.

There are two distinct cases:

1. *Points* specifies a single point. In this case, Text may be a single scalar character, a simple vector, or a matrix containing a block of text. The result is that the character, string, or matrix is written at the specified point.
2. *Points* specifies more than one point. There are three possibilities:
 1. If Text is a scalar, its contents are written at each of the points in *Points*. This means that by enclosing a vector or matrix, you can draw a string or block of text at several locations.
 2. If Text is a vector, each element of Text is written at the corresponding position in *Points*.
 3. If Text is a matrix, each row of Text is written at the corresponding position in *Points*.

FontObj specifies a **single** font to be used to write the Text. See a description of the *FontObj* property for details.

FCol specifies the colour of the Text. For a single text item, *FCol* may be a single number which specifies one of the standard MS-Windows colours, or a simple 3-element numeric vector of RGB colour intensities. If more than one text item is involved, *FCol* may be a vector which specifies the colour for each item separately. If so, its length must be the same as the number of points specified by *Points*.

BCol specifies the background colour of the text, that is, the colour for the part of the character cell that is blank. It is defined in the same way as *FCol*.

HAlign and *VAlign* specify the horizontal and vertical alignment of the text respectively. They may each be numeric scalars or vectors with the same length as the number of points specified in *Points*. See *HAlign* and *VAlign* for details.

When one or more of *FCol*, *BCol*, *HAlign* and *VAlign* are vectors, the different components of Text are drawn using the corresponding colours and alignments.

The value of the *Dragable* property specifies whether or not the Text object can be dragged by the user. The value of the *AutoConf* property determines whether or not the Text object is repositioned when its parent is resized.

EXAMPLES

Write 'A' at (10,20)

```
'g.t1' WC 'Text' 'A' (10 20)
```

Write 'h' at (10,20) in red

```
'g.t1' WC 'Text' 'h' (10 20) ('FCol' 255 0 0)
```

Write 'Hello' at (10,20)

```
'g.t1' WC 'Text' 'Hello' (10 20)
```

Write 'THIS IS A
BLOCK OF
TEXT' at (20,30)

```
BLK+3 9p 'THIS IS A BLOCK OF TEXT'  
'g.t1' WC 'Text' BLK (10 20)
```

Write 'A' at (10,20) and at (30,40)

```
'g.t1' WC 'Text' 'A' ((10 30)(20 40))
```

Write a red '+' at (10,20) and a green '+' at (20 40)

```
'g.t1' WC 'Text' '+' ((10 30)(20 40))('FCol' (255 0 0)(0 255 0))
```

Write 'Hello' at (10,20) and at (30,40)

```
'g.t1' WC 'Text' ('Hello') ((10 30)(20 40))
```

Write 'A' at (10,20) and 'B' at (30,40)

```
'g.t1' WC 'Text' 'AB' ((10 30)(20 40))
```

Write 'Hello' at (10,20) and 'World' at (30,40)

```
'g.t1' WC 'Text' ('Hello' 'World')((10 30)(20 40))
```

2.70 Timer Object

Purpose: To generate an action at regular intervals.

Parents ActiveXControl Animation Bitmap BrowseBox Button ButtonEdit Calendar Circle Clipboard Combo ComboEx CoolBand CoolBar Cursor DateTimePicker Edit Ellipse FileBox Font Form Grid Group HTMLRenderer Icon Image ImageList Label List ListView Locator Marker MDIClient Menu MenuBar MenuItem Metafile MsgBox NetClient NetType OLEClient OLEServer Poly Printer ProgressBar PropertyPage PropertySheet Rect RichEdit Root Scroll Separator SM Spinner Splitter Static StatusBar StatusField SubForm SysTrayItem TabBar TabBtn TabButton TabControl TCPSocket Text Timer TipField ToolBar ToolButton ToolControl TrackBar TreeView UpDown

Properties Type Interval Active Event Data KeepOnClose FireOnce MethodList ChildList EventList PropList

Methods Detach Wait

Events Close Create Timer

Description

The Timer object is used to generate an event at regular intervals. It can be used to produce animation and to implement "repeaters" such as spin buttons.

The *Interval* property specifies how often the Timer generates events and is defined in milliseconds. Its default value is 1000.

The *Active* property determines whether or not the Timer generates events and can be used to switch the Timer off and on as required.

The *FireOnce* property may be used to implement a once-off Timer event and has the value 0, 1 or 2.

Note that if you create a Timer object whose *Timer* event generates an error (for example by attaching it to a non-existent callback) it may be very difficult or even impossible to type into the Session, because the error will be displayed over and over again. Care is therefore recommended.

2.71 TipField Object

Purpose: To display pop-up help.

Parents ActiveXControl CoolBand Form Group OLEServer PropertyPage PropertySheet Root SubForm TCPSocket

Children Circle Ellipse Font Marker Poly Rect Text Timer

Properties Type Event FontObj FCol BCol Data Translate KeepOnClose MethodList ChildList EventList PropList

Methods Detach ChooseFont

Events Close Create FontOK FontCancel

Description

The TipField is used to display pop-up help when the user moves the mouse pointer over an object.

Most of the GUI objects supported by Dyalog APL/W have a *Tip* and a *TipObj* property. *TipObj* specifies the name of, or ref to, a TipField object, and *Tip* specifies a "help" message. The TipField automatically pops-up to display the *Tip* when the user moves the mouse pointer over the object. It disappears when the user moves the mouse pointer away.

The TipField is a simple box with a 1-pixel black border in which the text specified by *Tip* is displayed. *FCol*, *BCol* and *FontObj* can be used to customise the appearance of the text within the box. *FCol* specifies the colour of the text; *BCol* specifies the background colour with which the box is filled. The default is black on yellow.

If you wish to display *Tips* for particular objects in different fonts and colours, you must create a separate TipField for each combination of colour and font you need.

2.72 ToolBar Object

Purpose: To manage a group of controls such as *Button* s.

Parents ActiveXControl CoolBand Form SubForm

Children Bitmap BrowseBox Button Calendar Circle Combo ComboEx Cursor DateTimePicker Edit Ellipse FileBox Font Group Icon Image ImageList Label List ListView Locator Marker Menu Metafile MsgBox OCXClass Poly ProgressBar Rect RichEdit Scroll SM Spinner Static SubForm Text Timer TrackBar TreeView UpDown

Properties Type Posn Size Coord Align Border Active Visible Event VScroll HScroll Sizeable FontObj FCol BCol Picture OnTop IconObj CursorObj AutoConf YRange XRange Data Attach TextSize EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel Select

Description

The *ToolBar* object is used to display and manage a set of controls. It is typically used to present a set of *Buttons* which the user can press to perform various actions. However, the *ToolBar* has the ability to manage other controls too.

By default, the *ToolBar* is a raised bar stretched across the top of its parent form. You can alter its appearance using its *EdgeStyle* property and you can control its alignment with its *Align* property. *Align* can be set to *Top* (the default), *Bottom*, *Left* or *Right* and causes the *ToolBar* to be attached to the corresponding edge of the *Form*. A *ToolBar* aligned *Top* or *Bottom* will automatically stretch or shrink horizontally when its parent *Form* is resized, but it will remain fixed vertically. A *ToolBar* aligned *Left* or *Right* will stretch vertically but will remain fixed horizontally. By default a *ToolBar* occupies the entire width or length of the side of the *Form* to which it is attached and is 30 pixels high or wide. You can change these defaults using the *Posn* and *Size* properties.

A *ToolBar* organises its child controls in the order they are created. The way this is done is governed by the value of the *Align* property. If *Align* is *Top* or *Bottom*, the *ToolBar* arranges its controls in rows across the screen. If *Align* is *Left* or *Right*, the *ToolBar* arranges controls in columns.

The first control added to a *ToolBar* is automatically positioned 2 pixels down and 2 pixels across from its top left corner. The rule for positioning subsequent controls depends upon the value of the *Align* property.

If *Align* is '*Top*' or '*Bottom*', controls are positioned so as to be horizontally adjacent to one another. Whenever a control is added it is positioned relative to the one that immediately preceded it so that its top left corner meets the top right corner of the previous one. The *HScroll* property determines what happens when the end of the *ToolBar* is reached. If *HScroll* is 0 (the default) a control that would otherwise extend beyond the width of the *ToolBar* is instead positioned immediately below the first control in the *ToolBar*, thereby starting a new row. Note however that the *ToolBar* is not automatically resized vertically to accommodate a second row. If you want a multi-row *ToolBar* you have to set its height explicitly. If *HScroll* is *-1* or *-2*, controls continue to be added along the *ToolBar* even though they extend beyond its right edge and may be scrolled into view using a mini scrollbar. If *HScroll* is *-1*, the scrollbar is shown whether or not any controls extend beyond the width of the *ToolBar*. If *HScroll* is *-2*, the scrollbar appears only if required and may appear or disappear when the user resizes the parent *Form*.

If *Align* is 'Left' or 'Right', controls are positioned so as to be vertically adjacent to one another. Whenever a control is added, its top left corner is positioned against the bottom left corner of the previous control. The *VScroll* property determines what happens when the bottom of the ToolBar is reached. If *VScroll* is 0 (the default) a control that would otherwise extend beyond the bottom of the ToolBar is instead positioned immediately to the right of the first one; thereby starting a new column. Note however that the ToolBar is not automatically resized horizontally to accommodate a second column. You must set the width of the ToolBar explicitly.

If *VScroll* is -1 or -2, controls continue to be added down the ToolBar even though they extend beyond its bottom edge and may be scrolled into view using a mini scrollbar. If *VScroll* is -1, the scrollbar is shown whether or not any controls extend beyond the bottom of the ToolBar. If *VScroll* is -2, the scrollbar appears only if required and may appear or disappear when the user resizes the parent *Form*.

VScroll and *HScroll* may only be set when the object is created and may not subsequently be changed.

If you specify a value for its *Posn* property, a control will be placed at the requested position regardless of the value of *Style*, *VScroll* or *HScroll*. However, the next control added will take its default position from the previous one according to the value of these properties. Thus if you wish to group your controls together with spaces between the groups, you need only specify the position of the first one in each group.

The ToolBar object was introduced in Dyalog APL before an appropriate standard Windows control existed. The ToolBar object should be considered as a legacy object and used only in old GUI applications. The *ToolControl* object should be used instead.

2.73 ToolButton Object

Purpose: The ToolButton object represents a button in a ToolControl.

Parents ToolControl

Children Bitmap Timer

Properties Type Caption Posn Size Style State Active Visible Event ImageIndex Data Hint HintObj Tip Accelerator Popup KeepOnClose MethodList ChildList EventList PropList

Methods Detach

Events Close Create Help MouseDown MouseUp MouseMove MouseDbClick Select

Description

The *ToolButton* object represents a selectable button in a *ToolControl* object.

A *ToolButton* displays a text string, defined by its *Caption* property, and an image defined by its *ImageIndex* property. Apart from these characteristics, the appearance of a *ToolButton* is controlled by its parent *ToolControl* object.

ImageIndex is an index into an *ImageList* which contains a set of icons or bitmaps. The *ImageList* itself is named by the *ImageListObj* property of the parent *ToolControl*.

Typically, you will create up to three *ImageLists* as children of the *ToolControl*. These will be used to specify the pictures of the *ToolButton* objects in their normal, highlighted (sometimes termed *hot*) and inactive states respectively. The set of images in each *ImageList* is then defined by creating unnamed *Bitmap* or *Icon* objects as children. Finally, when you create each *ToolButton* you specify *ImageIndex*, selecting the three pictures which represent the three possible states of the button.

If you specify only a single *ImageList*, the *picture* on the *ToolButton* will be the same in all three states.

The behaviour and appearance of a *ToolButton* is further defined by its *Style* property, which may be 'Push', 'Check', 'Radio', 'Separator' or 'DropDown'.

Push buttons are used to generate actions and pop in and out when clicked. Radio and Check buttons are used to select options and have two states, normal (out) and selected (in). Their *State* property is 0 when the button is in its normal (unselected state) or 1 when it is selected.

A group of adjacent *ToolButtons* with *Style* 'Radio' defines a set in which only one of the *ToolButtons* may be selected at any one time. The act of selecting one will automatically deselect any other. Note that a group of Radio buttons must be separated from Check buttons or other groups of Radio buttons by *ToolButtons* of another *Style*.

A *ToolButton* with *Style* 'Separator' has no *Caption* or picture, but appears as a vertical line and is used to separate groups of buttons.

A *ToolButton* with *Style* 'DropDown' has an associated popup *Menu* object which is named by its *Popup* property. There are two cases to consider.

If the *ShowDropDown* property of the parent *ToolControl* is 0, clicking the *ToolButton* causes the popup menu to appear. In this case, the *ToolButton* itself does not itself

generate a *Select* event; you must rely on the user selecting a *MenuItem* to specify a particular action.

If the *ShowDropDown* property of the parent *ToolControl* is 1, clicking the dropdown button causes the popup menu to appear; clicking the *ToolButton* itself generates a *Select* event, but does not display the popup menu.

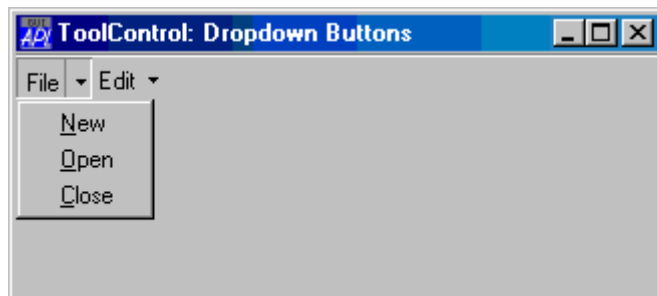
The following example illustrates the use of *DropDown* buttons:

```
'F'WC'Form' 'ToolControl: Dropdown Buttons'('Size' 20 40)
'F.TB'WC'ToolControl'('ShowDropDown' 1)

:With 'F.FMENU'WC'Menu' A Popup File menu
  'NEW'WC'MenuItem' '&New'
  'OPEN'WC'MenuItem' '&Open'
  'CLOSE'WC'MenuItem' '&Close'
:EndWith

:With 'F.EMENU'WC'Menu' A Popup File menu
  'CUT'WC'MenuItem' 'Cu&t'
  'COPY'WC'MenuItem' '&Copy'
  'PASTE'WC'MenuItem' '&Paste'
:EndWith

'F.TB.B1'WC'ToolButton' 'File'('Style' 'DropDown')('Popup' 'F.FMENU')
'F.TB.B2'WC'ToolButton' 'Edit'('Style' 'DropDown')('Popup' 'F.EMENU')
```



2.74 ToolControl Object

Purpose: The *ToolControl* object provides a standard Windows *ToolBar*.

Parents *ActiveXControl* *CoolBand* *Form* *SubForm*

Children Bitmap BrowseBox Button Combo ComboEx Cursor Edit FileBox Font Group Icon Image ImageList Label List ListView Locator Menu MenuBar Metafile MsgBox OCXClass ProgressBar RichEdit Scroll SM Spinner Static SubForm Timer ToolButton TrackBar TreeView UpDown

Properties Type Posn Size Style Align Visible Event ImageListObj FontObj Data Attach Handle KeepOnClose MultiLine Transparent Divider ShowCaptions ShowDropDown Dockable UndocksToRoot Redraw ButtonsAcceptFocus MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel DockStart DockMove DockRequest DockAccept DockEnd DockCancel

Description

The *ToolControl* object provides an interface to the native Windows *ToolBar* control and supersedes the Dyalog APL *ToolBar* object.

The tools on a *ToolControl* are normally represented by *ToolButton* objects, but the *ToolControl* may also act as a parent for other objects, including a *MenuBar* (see below).

Unlike the *ToolBar*, the *ToolControl* fully determines the positioning of its children automatically and this is governed by their order of creation. The *Posn* property of any child of a *ToolControl* is therefore read-only. Furthermore, the height of objects in a *ToolControl* may be no greater than that of a *ToolButton* in the same *ToolControl*. This in turn is governed by the sizes of the *FontObj* and *ImageList* in use in that *ToolControl*. *ToolControl* objects should be used in preference to *ToolBar* objects.

If a *ToolControl* is the child of a *Form*, its position and orientation is defined by its *Align* property. This property is ignored if the *ToolControl* is the child of a *CoolBand*.

The overall appearance of the *ToolButton* objects displayed by the *ToolControl* is defined by the *Style* property of the *ToolControl* itself, rather than by individual *ToolButtons*. This may be 'Buttons', 'FlatButtons', 'List' or 'FlatList'.

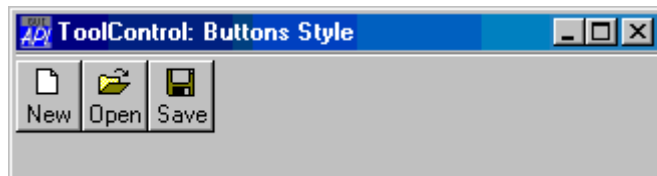
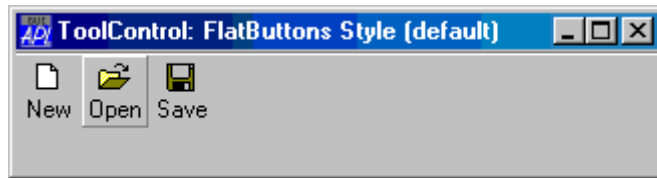

```

'F'WC'Form' 'ToolControl: FlatButtons Style (default)'('Size' 10 40)
'F.TB'WC'ToolControl'

'F.TB.IL'WC'ImageList'('Masked' 0)
'F.TB.IL.'WC'Bitmap'('Comctl32' 120)
'F.TB'WS'ImageListObj' 'F.TB.IL'

'F.TB.B1'WC'ToolButton' 'New'('ImageIndex' 7)
'F.TB.B2'WC'ToolButton' 'Open'('ImageIndex' 8)
'F.TB.B3'WC'ToolButton' 'Save'('ImageIndex' 9)

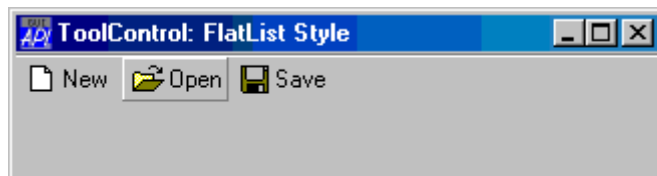
```



```

'F.TB'WC'ToolControl'('Style' 'Buttons')

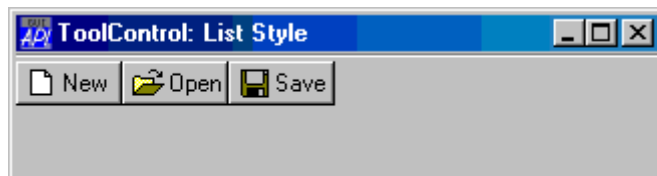
```



```

'F.TB'WC'ToolControl'('Style' 'FlatList')

```



```

'F.TB'WC'ToolControl'('Style' 'List')

```

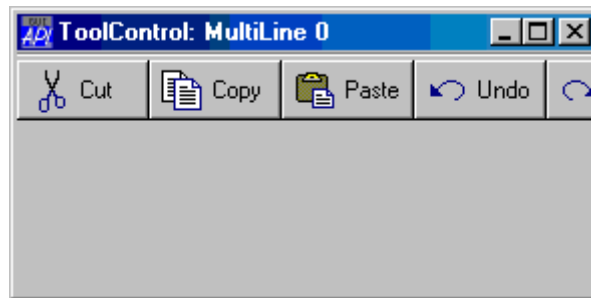
The presence or absence of a recessed line drawn above, below, to the left of, or to the right of the ToolControl is controlled by the *Divider* property whose default is 1 (show divider).

The *MultiLine* property specifies whether or not *ToolButtons* (and other controls) are arranged in several rows (or columns) when there are more than will otherwise fit. If *MultiLine* is 0 (the default), the *ToolControl* object clips its children and the user must resize it to bring more objects into view.

```
'F'WC'Form' 'ToolControl: MultiLine 0'('Size' 20 36)
'F.TB'WC'ToolControl'('Style' 'List')

'F.TB.IL'WC'ImageList'('Masked' 0)('Size' 24 24)
'F.TB.IL.'WC'Bitmap'('ComCtl32' 121)A STD_LARGE
'F.TB'WS'ImageListObj' 'F.TB.IL'

'F.TB.B1'WC'ToolButton' 'Cut'('ImageIndex' 1)
'F.TB.B2'WC'ToolButton' 'Copy'('ImageIndex' 2)
'F.TB.B3'WC'ToolButton' 'Paste'('ImageIndex' 3)
'F.TB.B4'WC'ToolButton' 'Undo'('ImageIndex' 4)
'F.TB.B5'WC'ToolButton' 'Redo'('ImageIndex' 5)
'F.TB.B6'WC'ToolButton' 'Delete'('ImageIndex' 6)
```



```
'F.TB'WC'ToolControl'('MultiLine' 1)('Style' 'List')
```

The *Transparent* property specifies whether or not the *ToolControl* is transparent. If so, the visual effect is as if the *ToolButtons* (and other controls) were drawn directly on the parent *Form* as illustrated below:

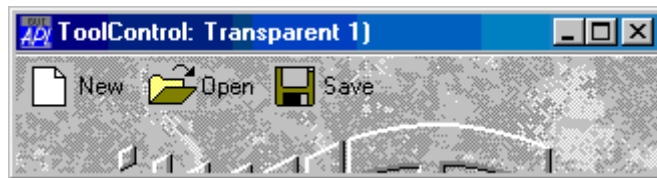
```

'F'⎕WC'Form' 'ToolControl: Transparent 1)'('Size' 10 40)
'F.BM'⎕WC'Bitmap' 'C:\WINDOWS\WINLOGO'
'F'⎕WS'Picture' 'F.BM' 1

'F.TB'⎕WC'ToolControl'('Transparent' 1)('Style' 'FlatList')
'F.TB.IL'⎕WC'ImageList'('Masked' 0)('Size' 24 24)
'F.TB.IL.'⎕WC'Bitmap'('ComCtl32' 121)⎕ STD_LARGE
'F.TB'⎕WS'ImageListObj' 'F.TB.IL'

'F.TB.B1'⎕WC'ToolButton' 'New'('ImageIndex' 7)
'F.TB.B2'⎕WC'ToolButton' 'Open'('ImageIndex' 8)
'F.TB.B3'⎕WC'ToolButton' 'Save'('ImageIndex' 9)

```



The *ShowCaptions* property specifies whether or not the captions of *ToolButton* objects are drawn. Its default value is 1 (draw captions). *ToolButtons* drawn without captions occupy much less space and *ShowCaptions* provides a quick way to turn captions on/off for user customisation.

The *ShowDropDown* property specifies whether or not a drop-down menu symbol is drawn alongside *ToolButtons* which have *Style* 'DropDown'. *ShowDropDown* also affects the behaviour of such *ToolButton* objects when clicked.

The *ButtonsAcceptFocus* property determines how the *ToolControl* responds to the Tab and cursor movement keys.

As a special case, the *ToolControl* may contain a *MenuBar* as its **only** child. In this case, Dyalog APL causes the menu items to be drawn as buttons as shown below.

Although nothing is done to prevent it, the use of other objects in a *ToolControl* containing a *MenuBar*, is not supported.

```

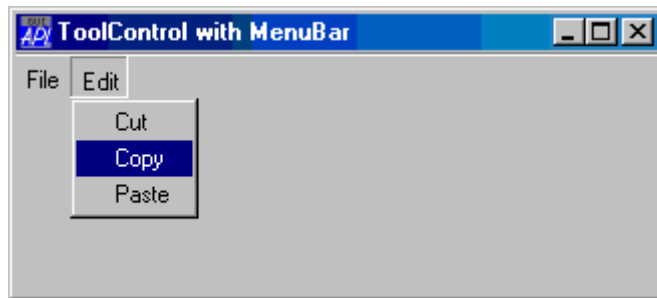
'F'WC'Form' 'ToolControl with MenuBar'('Size' 20 40)
'F.TB'WC'ToolControl'

:With 'F.TB.MB'WC'MenuBar'
  :With 'File'WC'Menu' 'File'
    'New'WC'MenuItem' 'New'
    'Open'WC'MenuItem' 'Open'
    'Close'WC'MenuItem' 'Close'
  :EndWith

  :With 'Edit'WC'Menu' 'Edit'
    'Cut'WC'MenuItem' 'Cut'
    'Copy'WC'MenuItem' 'Copy'
    'Paste'WC'MenuItem' 'Paste'
  :EndWith

:EndWith

```



2.75 TrackBar Object

Purpose: The TrackBar object is a slider control that allows the user to enter a value by positioning a pointer (thumb) on a scale.

Parents ActiveXControl CoolBand Form Grid Group PropertyPage SubForm ToolBar
ToolControl

Children Bitmap Circle Cursor Ellipse Font Icon Marker Poly Rect Text Timer

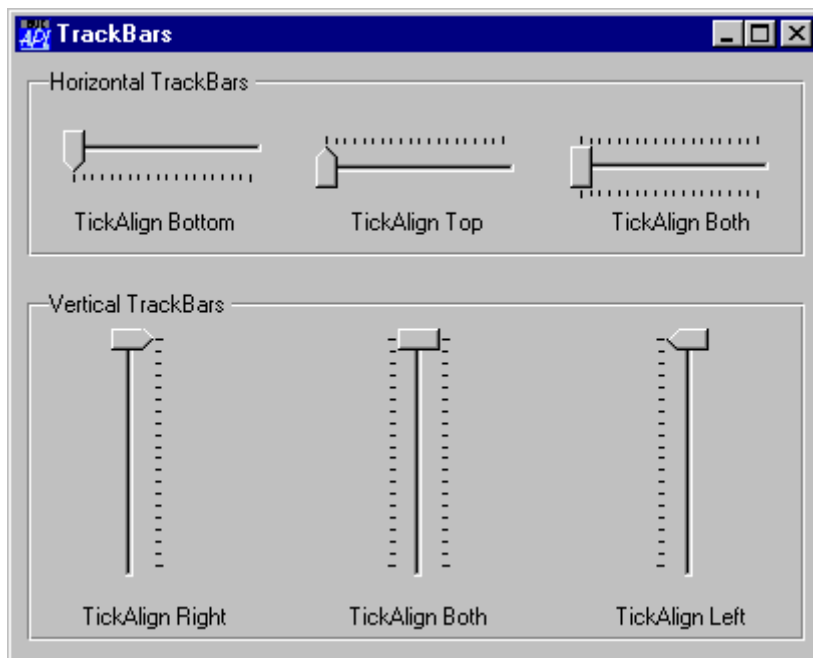
Properties Type Posn Size Style Coord Border Active Visible Event Thumb Step VScroll
HScroll Limits SelRange Sizeable Dragable FontObj BCol CursorObj AutoConf Data
Attach EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator TickAlign
TickSpacing HasTicks ShowThumb TrackRect ThumbRect AcceptFiles KeepOnClose
Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove Select ThumbDrag Scroll

Description

The `TrackBar` object consists of a window which contains a slider bar, a thumb, and a set of tick marks. 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 of five, the slider can only occupy six positions: a position at the left side of the `TrackBar` and one position for each increment in the range. Typically, each of these positions is identified by a tick mark. `TrackBars` can have either a vertical or horizontal orientation. They can have tick marks on either side, both sides, or neither. A selection of different `TrackBars` is illustrated below.



The position and size of the container window are defined by the `Posn` and `Size` properties. Its appearance is defined by the `EdgeStyle`, `Border` and `BCol` properties. The defaults are (`'EdgeStyle' 'None'`), (`'Border' 0`) and (`'BCol' 0`). The default background colour (`'BCol' 0`) obtains either the standard Window Background colour, or grey to match the colour of the parent object if it has a 3-dimensional appearance.

The orientation of a *TrackBar* is determined by the *HScroll* and *VScroll* properties. A horizontal *TrackBar* is obtained by setting *HScroll* to `-1` and *VScroll* to `0`. This is the default. A vertical *TrackBar* is obtained by setting *VScroll* to `-1` and *HScroll* to `0`.

VScroll and *HScroll* may only be set when the object is created and may not subsequently be changed.

The *ShowThumb* property determines whether or not the thumb is visible. Its default value is `1`. You may toggle this property dynamically using `WS`.

The *TrackBar* optionally displays tick marks at the two ends of the slider bar and spaced out along it. This behaviour is determined by the *HasTicks* property which may be `1` (the default) or `0` and may be set *only* when the object is created by `WC`.

If *HasTicks* is `1`, the position and frequency of the tick marks is determined by the *TickAlign* and *TickSpacing* properties. Note that *TickAlign* may only be set when the *TrackBar* is created with `WC` and may not be altered using `WS`.

The slider and tick marks in a horizontal *TrackBar* are drawn along the top of the enclosing window. The slider and tick marks in a vertical *TrackBar* are drawn along the left edge of the window. The position and size of the slider and the thumb may be obtained from the *TrackRect* and *ThumbRect* properties which report these values in pixels. These are *read-only* properties and cannot not be set with `WC` or `WS`.

The value of the *TrackBar* is determined by its *Thumb* property which is an *integer* that may be set with `WS` or retrieved with `WC`. The *Limits* property specifies the minimum and maximum values of *Thumb* corresponding to its position at the two ends of the slider bar. The *Step* property is a 2-element integer vector defining the small and large increments by which the *Thumb* moves. A small step is obtained by pressing a cursor movement key; a large step is achieved by clicking the left mouse button either side of the thumb or by pressing *Page Up* and *Page Down*. The user may also drag the thumb to a new position or move it directly to either end of the slider by pressing *Home* or *End*.

An alternative form of the *TrackBar* is obtained by setting the *Style* property to `'Selection'`. This may only be done when the object is created using `WC`. This style of *TrackBar* has a slider that is represented by a recessed thick white rectangle instead of a solid black line. Furthermore, you can select a range of values within the *TrackBar* by setting the *SelRange* property. This causes the *TrackBar* to display a solid blue bar within the white slider and to show the corresponding tick marks as small triangles. Note that there is no way for the user to change *SelRange* directly; you can only do this using `WS`.

In addition to the normal mouse events, the `TrackBar` generates a *Scroll* and *ThumbDrag* event. The *Scroll* event is the same event that is generated by a `Scroll` object and is reported when the user repositions the thumb. If enabled, the *ThumbDrag* event is reported continuously as the user drags the thumb with the mouse and may be used to synchronise the display of a corresponding value in another object.

2.76 TreeView Object

Purpose: The `TreeView` object displays a hierarchical list of items.

Parents `ActiveXControl` `CoolBand` `Form` `Group` `PropertyPage` `SubForm` `ToolBar` `ToolControl`

Children `Bitmap` `Circle` `Cursor` `Ellipse` `Font` `Icon` `ImageList` `Marker` `Poly` `Rect` `Text` `Timer`

Properties `Type` `Items` `Posn` `Size` `Coord` `Border` `Active` `Visible` `Event` `Depth` `HasLines` `HasButtons` `EditLabels` `ImageListObj` `ImageIndex` `SellImageIndex` `SellItems` `Sizeable` `Dragable` `FontObj` `FCol` `BCol` `CursorObj` `AutoConf` `Index` `Data` `Attach` `EdgeStyle` `Handle` `Hint` `HintObj` `Tip` `TipObj` `Translate` `Accelerator` `AcceptFiles` `KeepOnClose` `CheckBoxes` `FullRowSelect` `SingleClickExpand` `Redraw` `TabIndex` `AlwaysShowSelection` `MethodList` `ChildList` `EventList` `PropList`

Methods `Detach` `ChooseFont` `GetItemState` `SetItemState` `AddItems` `DeleteItems` `AddChildren` `DeleteChildren` `GetParentItem` `GetItemHandle` `SetItemImage` `ShowItem` `GetTextSize` `Animate` `GetFocus` `ShowSIP` `GetFocusObj`

Events `Close` `Create` `FontOK` `FontCancel` `DragDrop` `Configure` `ContextMenu` `DropFiles` `DropObjects` `Expose` `Help` `KeyPress` `GotFocus` `LostFocus` `MouseDown` `MouseUp` `MouseMove` `MouseDownClick` `MouseEnter` `MouseLeave` `MouseWheel` `GestureZoom` `GesturePan` `GestureRotate` `GestureTwoFingerTap` `GesturePressAndTap` `BeginEditLabel` `EndEditLabel` `Expanding` `Retracting` `ItemDown` `ItemUp` `ItemDbClick` `GetTipText` `Select`

Description

A `TreeView` object 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 bitmapped image, and each item can have a list of sub-items associated with it. By clicking an item, the user can expand and collapse the associated list of sub-items.

The contents of a `TreeView` object are defined by the *Items* property; a vector of character vectors that specifies the item labels.

The *ImageListObj*, *ImageIndex* and *SellImageIndex* properties define bitmapped images corresponding to each item. The bitmapped images are drawn to the left of the item labels.

ImageListObj specifies the name of a single *ImageList* object that contains one or more bitmaps.

ImageIndex and *SellImageIndex* are $\square$ IO sensitive scalars, or vectors with the same length as the number of items in the object. The value in the *i*th element specifies the image for the *i*th item and is an index into the corresponding *ImageList* object. *ImageIndex* specifies the image displayed when an item is *not selected*, *SellImageIndex* specifies the image displayed when an item is *selected*.

If *ImageListObj* is specified, but *ImageIndex* is empty or not specified, the first bitmap in the *ImageList* is drawn alongside every item. If an element of *ImageIndex* or *SellImageIndex* specifies a value that does not correspond to a bitmap in the *ImageList*, no picture is drawn.

The structure of the items (that is, the parent/child relationships of the items) is defined by the *Depth* property. This is either a scalar 0 (the default) which means that all items are *root* items, or it is a numeric vector of the same length as *Items*. Non-zero values in *Depth* indicate child items.

The *HasLines* property is 0, 1 or 2 and determines whether or not lines are drawn that link child items to their corresponding parent item. If *HasLines* is 0, no lines are drawn. If *HasLines* is 1, lines are drawn at all except the top level, that is, the object does not link items at the root of the hierarchy. The default value for *HasLines* is 2 which provides lines at all levels including the root.

The *HasButtons* property determines whether or not the *TreeView* object has a button to the left side of each parent item. It is Boolean with a default value of 1. The user can click the button to expand or collapse the child items as an alternative to double-clicking the parent item. Note that by itself, setting *HasButtons* to 1 does not add buttons to items at the root of the hierarchy. To achieve this you must also set *HasLines* to 2.

The *CheckBoxes* property specifies whether or not check boxes are displayed alongside items in a *TreeView*.

The *FullRowSelect* property specifies whether just the item itself, or the entire row of the *TreeView*, is highlighted when an item is selected. *FullRowSelect* should not be used if *HasLines* is 1 or 2

When the user presses the left mouse button over an item, the object generates an *ItemDown* event. This is followed by an *ItemUp* event when the mouse button is released. The object also generates an *ItemDbClick* event when the left mouse is double-clicked over an item. If all three events are enabled, they are reported in the order *ItemDown*, *ItemDbClick*, *ItemUp*.

When a parent item is in its retracted state (its children are not visible) it can be expanded to show its children by the user double-clicking its label or by clicking over its button or tree lines. An *Expanding* event is reported immediately before the children are shown. Similarly, when a parent item is in its expanded state, it can be retracted to hide its children when a *Retracting* event is reported. You can use the *Expanding* event to define new children for the object just before they are shown. You can also control the actions of these events using callback functions.

The *EditLabels* is a Boolean property (default 0) that determines whether or not the user may edit the labels which are specified by the *Items* property.

The *SelItems* property is a Boolean vector that indicates which of the items is currently selected and has the focus. If more items are visible than can fit within the object, a scrollbar is automatically provided. The *Index* property is a ± 10 sensitive integer that reports the index number of the first item displayed in the object and changes as the items are scrolled.

Warning

Due to the limitations of the Win32 TreeView object, it is necessary to query the state of each item in a TreeView in order to obtain the value of the *SelItems* property, making it a comparatively slow operation if there are a lot of *Items*.

2.77 UpDown Object

Purpose: The UpDown object is a pair of arrow buttons used to increment or decrement a value.

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm ToolBar ToolControl

Children Bitmap Circle Cursor Ellipse Font Icon Marker Poly Rect Text Timer

Properties Type Posn Size Coord Border Active Visible Event Thumb Step VScroll HScroll Wrap Limits Sizeable Dragable FCol BCol CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help MouseDown MouseUp MouseMove MouseEnter MouseLeave Select Spin

Description

An UpDown object is a pair of arrow buttons that the user can click to increment or decrement a value, such as a scroll position or a number displayed in a companion control. The *Spinner* object is actually a composite object consisting of an UpDown and a companion Edit.

3 Properties

3.1 APLVersion Property

Applies To: *Root*

Description

This is a read-only property that provides information about the Version of Dyalog APL that you are using. It is a 4-element vector of character vectors as described in the table below. In future releases these values may change, be removed, or new ones added.

Index	Description	Possible Values
[1]	Target Environment	Windows Windows-64 Windows Mobile Linux Linux-64 AIX AIX-64 Mac-64 Solaris Solaris-64
[2]	Version Number	
[3]	Version Type	w Windows s Server (terminal) version m Motif
[4]	Program Type	Development Runtime DLL DLLRT

EXAMPLE

'. '␣WG'APLVersion'			
Windows-64	18.0.38434.0	W	Development

3.2 Accelerator Property

Applies To

Description

This property specifies a keystroke that, when pressed by the user, will generate a *Select* event on an object. It applies to all objects whether or not they possess a "natural" *Select* event. You can therefore associate a keystroke with an arbitrary action on any object you desire.

The Accelerator property is a 2-element integer vector. The first element is a key number which is the number by which Windows *knows* the key. The second element is the shift state which is the sum of 1 (Shift key), 2 (Control key) and 4 (Alt key).

For example, to attach the keystroke Ctrl+A to an object, you would set its Accelerator to (65 2). To attach the keystroke Shift+Ctrl+F1 (key number 112), you would set its Accelerator to (112 3). Key numbers may be obtained by displaying the messages generated by the *KeyPress* event.

Note that a keystroke used as an Accelerator will **not** generate a *KeyPress* event.

3.3 AcceptFiles Property

Applies To

Description

The AcceptFiles property is Boolean and specifies whether or not an object will accept a file drag/drop operation. Its default value is 0. If set to 1, the object will report a *DropFiles* event when file icons are dropped on it.

3.4 Active Property

Applies To

Description

This property specifies whether or not an object is currently responsive to user actions. It is a single number with the value 0 (object is inactive and does not generate events) or 1 (object is active and capable of generating events). The default is 1.

Setting Active to 0 disables the object (and all its children), even though the object may be referenced in the argument to *Dyalog APL Language: Dq*. It is therefore possible to deactivate an object from a callback function.

In general, the text associated with an object whose Active property is 0 is displayed in grey.

3.5 Align Property

Applies To

Description

For an *Animation*, the Align property may be 'None' or 'Centre' ('Center'). If Align is 'None', the *Animation* window is automatically resized to fit the AVI being played. If Align is 'Centre', the AVI is centred in the *Animation* window. If the window is too small, the AVI is clipped.

For a *Button*, *Menu*, or *MenuItem* the Align property may be 'None', 'Left' or 'Right'. If the *Button* Style is 'Radio' or 'Check' this property specifies the position of the text relative to the button symbol. The default is 'Right'. For a *Button* with Style 'Push', the value of Align is 'None'.

For a *Button* with Style 'Radio' or 'Check' that is created as a child of a Grid the value of the Align property may also be 'Centre' or 'Center'. Either of these values causes the symbol part of the *Button* (the circle or checkbox) to be centred within the corresponding Grid cell(s).

For a *DateTimePicker*, the Align property specifies the horizontal alignment of the drop-down Calendar which may be 'Left' (the default) or 'Right'. This applies only if the Style of the *DateTimePicker* is 'Combo'.

For a single-line *Edit*, Align specifies the vertical alignment of the text. It may be 'None' (the default), 'Centre' or 'Center'.

For a *Menu*, *MenuItem*, or *StatusField*, Align 'Right' is used to position the object at the right end of its parent *MenuBar* or *StatusBar*. 'None' is equivalent to 'Left' which is the default.

For objects of type *CoolBar*, *Splitter*, *Scroll*, *StatusBar*, *TabBar*, *ToolBar* and *ToolControl*, Align may be 'None', 'Top', 'Bottom', 'Left' or 'Right'. It specifies to which (if any) of the four sides of the parent the object is anchored and also the default position and size of the object. Specifying Align typically causes the *Attach* property to be set to appropriate values as follows :

Align	Attach
'Top'	'Top' 'Left' 'Top' 'Right'
'Bottom'	'Bottom' 'Left' 'Bottom' 'Right'
'Left'	'Top' 'Left' 'Bottom' 'Left'
'Right'	'Top' 'Right' 'Bottom' 'Right'

These settings cause the object to remain at a fixed distance (in pixels) from the corresponding edge of the parent. Furthermore, the object will have a fixed height or width, but its length will stretch and shrink as the *Form* is resized.

Note that this does not apply to a *TabControl* for which the default value of Attach is 'None' 'None' 'None' 'None', regardless of the value of Align.

The default value of Align is 'Right' for a vertical *Scroll*, 'Bottom' for a horizontal *Scroll*, and 'Top' for a *CoolBar*, *TabBar*, *TabControl*, *ToolBar* and *ToolControl*. Furthermore, unless *Posn* and *Size* are specified explicitly, the object is placed along the corresponding edge of its parent.

For a *Scroll* object, Align also determines the direction of a *Scroll* object unless it is overridden by setting *HScroll* or *VScroll* directly. If neither *HScroll* or *VScroll* is defined and Align is 'Top' or 'Bottom', a horizontal scrollbar is provided. If neither *HScroll* or *VScroll* is defined and Align is 'None', 'Left' or 'Right', a vertical scrollbar is provided.

Note

The value of the Align property may **only** be assigned by *Dyalog APL Language*: *Wc* and may **not** be changed using *Dyalog APL Language*: *Ws*.

3.6 AlignChar Property

Applies To: *Grid*

Description

The AlignChar property specifies a character on which the data displayed in a column of a Grid is to be aligned vertically. It is useful to align columns of numbers that are formatted by the *FormatString* property. AlignChar may be a scalar or singleton that applies to all columns of the Grid, or a vector with one element per column.

If the data in the column is left-justified, it is aligned using the first occurrence of the alignment character in each cell counting from the left. If the data is right-justified, it is aligned using the first occurrence of the alignment character from the right-hand end of the text.

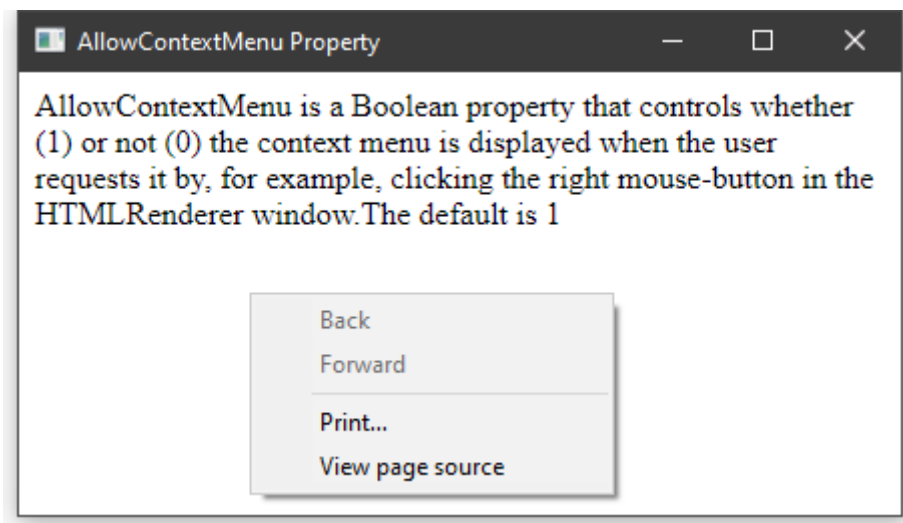
If the text in a cell does not contain an alignment character, it is aligned as if the alignment character were placed following the last digit.

3.7 AllowContextMenu Property

Applies To: *HTMLRenderer*

Description

This is a Boolean property that controls whether (1) or not (0) the context menu is displayed when the user requests it. The default is 1.



3.8 AlphaBlend Property

Applies To: *Form*

Description

The AlphaBlend property specifies a level of translucency which allows the area behind a *Form* to show through.

AlphaBlend is a scalar integer value in the range 0 to 255.

A value of 255 (the default) specifies no translucency, and the *Form* is entirely opaque obliterating anything behind it.

A value of 0 specifies total translucency and the *Form* itself is not visible. Furthermore, mouse events over the *Form* will not be reported by the *Form* itself but will be passed to any other windows underneath the *Form*.

Values in between specify varying levels of translucency.

3.9 AlwaysShowBorder Property

Applies To: *Grid*

Description

The AlwaysShowBorder property specifies whether or not the border around the current cell in a *Grid* is displayed when the *Grid* loses the focus.

It is a Boolean value with a default value of 1.

3.10 AlwaysShowSelection Property

Applies To: *Grid*, *ListView*, *TreeView*

Description

The AlwaysShowSelection property specifies whether or not the selection remains highlighted when the object loses the focus.

It is a Boolean value with a default value of 1.

If `AlwaysShowSelection` is 1, the highlight is dimmed. If `AlwaysShowSelection` is 0, the highlight disappears.

3.11 ArcMode Property

Applies To: *Circle, Ellipse*

Description

This property determines how arcs are drawn. Its value is 0, 1 or 2.

0	only the arc is drawn
1	arcs define "arc segments", with a single straight line joining the two ends of the arc together
2	arcs define "pie segments", with lines drawn from the start and end points of the arc to the centre

Note that the segments defined by ArcMode 1 and 2 may be filled (by setting *FStyle*).

3.12 Array Property

Applies To: *Clipboard*

Description

This property may be used to set or retrieve the contents of the Windows clipboard as a Dyalog APL array.

3.13 AsChild Property

Applies To: *HTMLRenderer*

Description

The `AsChild` property is a Boolean (default 0) indicating how the *HTMLRenderer* object is displayed. `AsChild` must be set when the object is created and may not subsequently be changed.

If `AsChild` is 0 (the default) the *HTMLRenderer* is displayed in a separate top-level window. If the *HTMLRenderer* is created (with `AsChild` 0) as a child of another object it

still appears as a separate window and its *Size* and *Posn* relate to the screen rather than to its parent object. However, it is a member of its parent object's hierarchy and will disappear when its parent is closed.

If *AsChild* is 1, the *HTMLRenderer* must be created as a child of a valid parent type other than *Root* (which is not supported) and is displayed in a sub-window within its parent.

This property only applies to Microsoft Windows. On other platforms it is ignored.

3.14 Attach Property

Applies To

Description

This property specifies how an object responds to its parent being resized. It is a 4-element vector of character vectors which defines how each of the four edges of the object moves in response to a resize request made by the parent. Note that this property is only effective if the value of *AutoConf* on the parent is 2 or 3 and *AutoConf* for the object itself is 1 or 3.

The 4 elements of *Attach* refer to the Top, Left, Bottom and Right edges of the object respectively. Their values may be defined as follows :

Element	Value	Meaning
[1]	'Top'	top edge of object attached to <i>top</i> edge of parent.
	'Bottom'	top edge of object attached to <i>bottom</i> edge of parent.
	'None'	top edge of object is not attached to its parent
[2]	'Left'	left edge of object is attached to <i>left</i> edge of parent
	'Right'	left edge of object is attached to <i>right</i> edge of parent
	'None'	left edge of object is not attached to its parent
[3]	'Top'	bottom edge of object is attached to <i>top</i> edge of parent.
	'Bottom'	bottom edge of object is attached to <i>bottom</i> edge of parent.
	'None'	bottom edge of object is not attached to its parent
[4]	'Left'	right edge of object is attached to <i>left</i> edge of parent
	'Right'	right edge of object is attached to <i>right</i> edge of its parent
	'None'	right edge of object is not attached to its parent

If an edge of the object is attached to an edge of its parent, its position in absolute (pixel) terms remains fixed relative to that edge when its parent is resized. Thus if *Coord* is 'Pixel', the corresponding *Posn* or *Size* property of the object remains unaffected by the resize. If *Coord* has any other value, the value of *Posn* or *Size* will change.

If an edge of the object is *not* attached to its parent, its absolute position (in pixels) will change in proportion to the size change (in the corresponding direction) of its parent. Thus if *Coord* is 'Pixel', the corresponding *Posn* or *Size* property of the object will change as a result of the resize. If *Coord* has any other value, the value of *Posn* or *Size* will be unaffected.

The default value of *Attach* is ('None' 'None' 'None' 'None'). This causes the object to reposition and resize itself in proportion to its parent.

Some objects have an *Align* property which, among other things, provides a quick way to set their *Attach* property. Examining this mechanism may help to further explain how the *Attach* property works. Setting *Align* to 'Top' has the effect of setting *Attach* to ('Top' 'Left' 'Top' 'Right'). Attaching the top edge of the object to the top edge of its parent causes the object to remain at a fixed distance from the top edge of its parent. The additional measure of attaching its bottom edge to the top edge of its parent causes the height of the object to remain fixed. Attaching the left and right edges of the object to the corresponding edges of its parent causes the object to shrink

and expand as the parent is resized horizontally. If you position the object at (0 0) and set its width to be the same as the width of its parent, you have an object that always occupies the entire length of its parent, yet remains of fixed height. This is precisely the behaviour required for a *ToolBar* or a top *Scroll Bar*. For further details, see *Align* property.

3.15 AutoArrange Property

Applies To: *ListView*

Description

The *AutoArrange* property is Boolean and specifies whether or not the items in a *ListView* object are automatically re-arranged when a single item is repositioned. Its default value is 0.

3.16 AutoBrowse Property

Applies To: *OLEClient*

Description

This property is retained for backwards compatibility with previous versions of Dyalog APL, but is no longer relevant. Setting it has no effect.

3.17 AutoConf Property

Applies To

Description

This property determines what happens to an object when its **parent** is resized, and how resizing an object affects its children. It may take one of the following values; the default is 3.

0	Ignore resize by parent. Do not propagate resize to children.
1	Accept resize by parent. Do not propagate resize to children.
2	Ignore resize by parent. Propagate resize to children.
3	Accept resize by parent. Propagate resize to children.

If AutoConf is 0 or 2, the object's **physical** size (in pixels) and position (in pixels) relative to the top left corner of its parent remains unchanged when its parent is resized. If the object has 'Prop' or 'User' co-ordinates, the values of its *Posn* and *Size* properties will change as a result.

If AutoConf is 1 or 3, by default, the object is physically reconfigured when its parent is resized such that its **relative** size and position within its parent remain unchanged. If the object has 'Pixel' co-ordinates, the values of its *Posn* and *Size* properties will change as a result. Note that this default processing can be prevented by inhibiting the *Configure* (31) Event.

If AutoConf is 0 or 1 and the object is resized, either by its parent or directly by the user, it does **not** attempt to physically reconfigure its children. This means that if the children have 'Prop' or 'User' co-ordinates, the values of their *Posn* and *Size* co-ordinates will change as a result.

If AutoConf is 2 or 3 and the object is resized, either by its parent or directly by the user, it propagates a *Configure* (31) Event to each of its children. By default this means that the object's children will be physically reconfigured so that they maintain their relative positions and sizes within it. If their co-ordinate system is 'Pixel', the values of their *Posn* and *Size* properties will change as a result.

Note

Additional or alternative control may be imposed by inhibiting the *Configure* (31) Event. This can be done either by setting the event's "action" code to `-1` or by returning a 0 from a callback function attached to it.

3.18 AutoExpand Property

Applies To: *Grid*

Description

This property is a 2-element Boolean value that specifies whether or not rows and columns may be added to a *Grid* object by the user.

If the first element of AutoExpand is 1, a row is added when the current cell is within the last row of the *Grid* and the user presses Cursor Down

Similarly, if the second element is 1, a column is added when the current cell is within the last column of the *Grid* and the user presses Cursor Right.

Note that when a row or column is added, the appropriate properties (including *Values* and *CellTypes*) are expanded accordingly.

The default value for *AutoExpand* is (0 0).

If *AutoExpand* is enabled, the *Grid* generates *AddRow* and *AddCol* events. You can return a zero from a callback function to selectively prevent the addition of rows and columns if appropriate.

3.19 AutoPlay Property

Applies To: *Animation*

Description

Specifies whether or not an AVI clip is played immediately when loaded in an *Animation* object.

AutoPlay is a single number with the value 0 (the default) or 1. If *AutoPlay* is 1, the AVI clip is automatically played through once from beginning to end when loaded from a file by the *AnimOpen* method.

3.20 BCol Property

Applies To

Description

This property defines the background colour(s) of an object.

For objects with more than one constituent part, it may specify a set of background colours, one for each part. A single colour is represented by a single number which refers to a standard colour, or by a 3-element vector which defines a colour explicitly in terms of its red, green and blue intensities.

If *BCol* is set to 0 (which is the default) the background colour is defined by your current colour scheme for the object in question. For example, if you select yellow as your MS-Windows "Menu Bar" colour, you will by default get a yellow background in *Menu* and *MenuItem* objects, simply by not specifying *BCol* or by setting it to 0.

If BCol is set to 0 (Zilde), Dyalog APL will never paint the background of the object. If therefore the object is overlaid by another window and then exposed, its background will not be redrawn and it will simply contain whatever was previously shown on that area of the screen.

A negative value of BCol refers to a standard MS-Windows colour as described below. Positive values are reserved for a possible future extension.

BCol	Colour Element	BCol	Colour Element
0	Default	-11	Active Border
-1	Scroll Bars	-12	Inactive Border
-2	Desktop	-13	Application Workspace
-3	Active Title Bar	-14	Highlight
-4	Inactive Title Bar	-15	Highlighted Text
-5	Menu Bar	-16	Button Face
-6	Window Background	-17	Button Shadow
-7	Window Frame	-18	Disabled Text
-8	Menu Text	-19	Button Text
-9	Window Text	-20	Inactive Title Bar Text
-10	Active Title Bar Text	-21	Button Highlight

If BCol contains a 3-element vector, it specifies the intensity of the red, green and blue components of the colour as values in the range 0-255. For example, (255 0 0) is red and (255 255 0) is yellow.

Note that the colour realised depends upon the capabilities of the display adapter and driver, and the current Windows colour map.

For a *Button*, BCol is only effective if the *Style* is 'Radio' or 'Check' and is ignored if the *Style* is 'Push'.

It is recommended that you only use **pure** background colours in *Combo* and *Edit* objects. This is because the text written in these objects cannot itself have a dithered background.

For the *Ellipse*, *Poly* and *Rect* objects, BCol specifies the background colour of the line drawn around the perimeter of the object and is effective only when a non-solid line

(*LStyle 1-4*) is used. It also specifies the colour used to fill the spaces between hatch lines if a hatch fill (*FStyle 1-6*) is used.

3.21 BandBorders Property

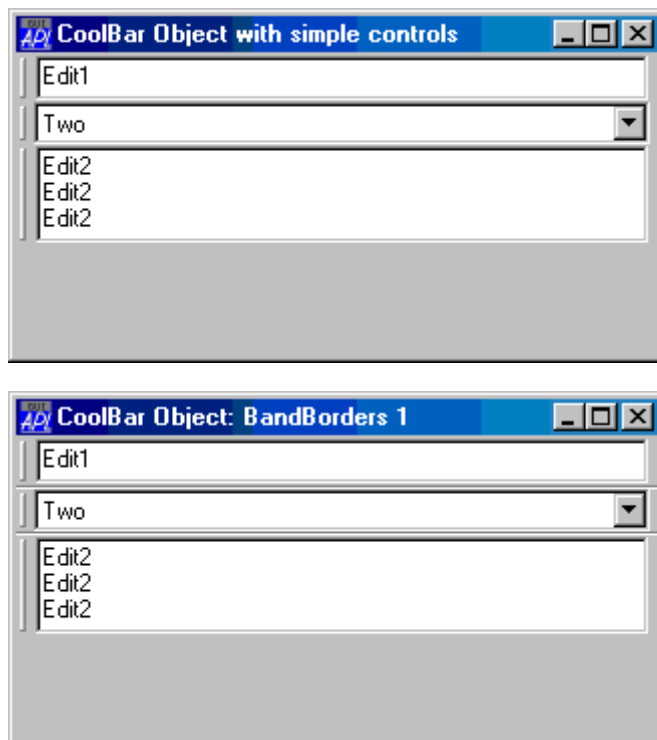
Applies To: *CoolBar*

Description

The *BandBorders* property specifies whether or not narrow lines are drawn to separate adjacent bands in a *CoolBar*.

BandBorders is a single number with the value 0 (no lines) or 1 (lines are displayed); the default is 0.

The effect of *BandBorders* is illustrated below.



3.22 BaseClass Property

Applies To: *NetType*

Description

This property specifies the name of the .NET class upon which the NetType is based. The newly created NetType inherits the properties, methods and events of this class.

3.23 Bits Property

Applies To: *Bitmap*, *Clipboard*, *Cursor*, *Icon*

Description

This property defines the pattern in a *Bitmap*, *Cursor*, or *Icon* object, or the pattern of a bitmap stored in the Windows clipboard.

For a *Bitmap*, *Clipboard* or *Icon*, Bits is an integer matrix each of whose elements represents the colour of the corresponding pixel in the bitmap. The colours are specified as 0-origin indices into the *CMap* property, which itself defines the complete set of different colours (the colour map) used by the object.

Please note that Bits and *CMap* may **only** be used to represent an image with a colour palette of **256 colours or less**. If the colour palette is larger, the values of Bits and *CMap* reported by `WMG` will be (0 0). For a high-colour image, use *CBits* instead.

For a *Cursor*, Bits is a Boolean matrix which specifies the shape of the cursor. For a *Cursor* and *Icon*, Bits is used in conjunction with the *Mask* property.

See *CMap* for further details.

3.24 Border Property

Applies To

Description

This property specifies whether or not an object is displayed with a border around it. It is a single number with the value 0 (no border) 1 (Border), or 2. The value 2 applies only to a *Form* and is used in combination with ('EdgeStyle' 'Dialog') to obtain standard dialog box appearance

For a *Form* or *SubForm*, the value of the Border property is only relevant if *Sizeable*, *Moveable*, *SysMenu*, *MaxButton* and *MinButton* are **all** 0.

Note

The value of the `Border` property may only be assigned by *Dyalog APL Language: Wc* and may **not** be changed using *Dyalog APL Language: Ws*.

3.25 BrowseFor Property

Applies To: *BrowseBox*

Description

The `BrowseFor` property is a character vector that specifies the type of resource to be the target of an object.

`BrowseFor` may be `'Directory'` (the default), `'File'`, `'Computer'` or `'Printer'`.

3.26 BtnPix Property

Applies To: *Button*, *Menu*, *MenuItem*

Description

This property is used to customise the appearance of a *Button*, *Menu* or *MenuItem*. It specifies the names of or refs to up to 3 *Bitmap* objects to be used to display the object under different circumstances. In general, `BtnPix` is a 3-element vector of character vectors or refs. However, if it defines a single *Bitmap*, it may be a single ref, a simple character scalar or vector, or an enclosed character vector.

The first *Bitmap* is displayed when the object is shown in its normal state. For a *Button*, this is when its `State` is 0. The second *Bitmap* is used for a *Menu* or *MenuItem*, when the object is selected (highlighted), or for a *Button* when its `State` is 1. The third *Bitmap* is used when the object is disabled by having its `Active` property set to 0.

For a *Button* with `Style` `'Push'`, this means that when the user clicks the *Button*, its appearance switches from the first to the second *Bitmap*, and then back again. To maintain the standard 3-D appearance, the *Bitmaps* should contain the correct shadow lines around their edges. For *Buttons* with `Style` `'Radio'` or `'Check'`, the *Button* will display one or other of the two *Bitmaps* according to its current `State`.

For example, to have a *Button* that displays a "Tick" or a "Cross" according to its `State`:

```
'YES' [WC 'Bitmap' 'C:\WDYALOG\YES.BMP'
'NO' [WC 'Bitmap' 'C:\WDYALOG\NO.BMP'
'f1.r1' [WC 'Button'('Style' 'Check')
          ('BtnPix' 'YES' 'NO')
```

3.27 Btns Property

Applies To: *MsgBox*

Description

The Btns property determines the set of buttons to be displayed in a *MsgBox*. It is a simple vector (one button) or a matrix with up to 3 rows, or a vector of up to 3 character vectors specifying the captions for up to 3 buttons. The buttons are arranged along the bottom of the dialog box in the order specified.

Under Windows, there are restrictions on these buttons. However the property has been designed more generally to be useful under different GUIs and perhaps later revisions of Windows.

Under Windows, the Btns property may specify one of **six** sets of buttons as follows.

- 'OK'
- 'OK' 'CANCEL'
- 'RETRY' 'CANCEL'
- 'YES' 'NO'
- 'YES' 'NO' 'CANCEL'
- 'ABORT' 'RETRY' 'IGNORE'

If any other combination is specified, *Dyalog APL Language: Wc* and *Dyalog APL Language: Ws* will report a DOMAIN ERROR. The names of the buttons are however case-insensitive, so the system will accept 'ok', 'Ok', 'oK' or 'OK'.

If the Btns property is not specified, it assumes a default according to *Style* as follows :

Style	Btns
'Msg' or 'Info'	'OK'
'Warn' or 'Error'	'OK' 'CANCEL'
'Query'	'YES' 'NO'

If *Style* is not specified, *Btns* defaults to 'OK'.

3.28 ButtonsAcceptFocus Property

Applies To: *ToolControl*

Description

This is a Boolean property that determines how the Tab key and other cursor movement keys are handled by a *ToolControl* object.

If *ButtonsAcceptFocus* is 0 (the default), when the user presses Tab or Shift+Tab to switch the input focus from another object to the *ToolControl*, the first *ToolButton* in the *ToolControl* receives the input focus and is highlighted. Pressing Tab or Shift+Tab again causes the input focus to move to another control. The cursor movement keys have no effect.

If *ButtonsAcceptFocus* is 1, when the user presses Tab or Shift+Tab to switch the input focus from another object to the *ToolControl*, the first or last *ToolButton* in the *ToolControl* receives the input focus and is highlighted. Note that the behaviour of Shift+Tab in this case is different. Pressing Tab or Shift+Tab again causes the input focus to move to another control, although if there is no other control to accept the input focus, it moves to the first or last *ToolButton* as appropriate. Pressing the cursor movement keys causes the input focus to move from one *ToolButton* to the next.

3.29 CBits Property

Applies To: *Bitmap*, *Clipboard*, *Icon*

Description

The *CBits* property represents the pixels that make up a picture..

CBits provides an alternative representation to that provided by the *Bits* and *CMap* properties which apply only to images with 256 colours or under. CBits may be used to represent both low-colour and high-colour images.

CBits is a rank-2 numeric array whose dimensions represent the rows and columns of pixels in the image. The values in CBits represent the colour of each pixel and (for an *Icon*) its transparency.

For a *Bitmap*, the colour value of each pixel is obtained by encoding the red, green and blue components, that is:

```
PIXEL ← 256 * RED + GREEN + BLUE
```

where RED, GREEN and BLUE are numbers in the range 0-255.

3.30 CEFVersion Property

Applies To: *HTMLRenderer*

Description

CEFVersion is a read-only property that reports the version of the [Chromium Embedded Framework \(CEF\)](#) that is being used. This information is primarily used for debugging and support.

It is a 10-element vector containing the following:

Index	Description
[1]	Formatted CEF release number. This is the primary identifier for a version of CEF.
[2]	CEF major version.
[3]	Commit number.
[4]	Chromium version number.
[5]	Chromium version number.
[6]	Chromium version number.
[7]	Chromium version number.
[8]	GIT Hashes
[9]	GIT Hashes
[10]	GIT Hashes

3.31 CMap Property

Applies To: *Bitmap, Clipboard, Cursor, Icon*

Description

This property defines the table of colours (the colour map) used by a *Bitmap* or *Icon* object or by a bitmap stored in the Windows clipboard. Its value is a 3-column integer matrix of numbers in the range 0-255. Each row represents a separate colour which is indexed (0-origin) by values in the *Bits* property. The 3 columns refer to the intensities of the red, green and blue components of colour respectively.

Please note that *Bits* and CMap may **only** be used to represent an image with a colour palette of **256 colours or less**. If the colour palette is larger, the values of *Bits* and CMap reported by `WMG` will be (0 0). For a high-colour image, use *CBits* instead.

When you create a *Bitmap* or *Icon* by specifying *Bits* and CMap, the actual colours you obtain are not necessarily those that you specified. This is partly due to hardware restrictions and partly due to the way in which Windows manages colours. Firstly, your display adapter and driver limit the number of pure colours that can be displayed at any one time and therefore define a maximum size for the colour map. For example, on a **standard** VGA you are limited to 16 different pure colours (additional ones are provided by **dithering**).

Secondly, Windows reserves a certain number of colours in the colour map for its own use. When an application requests a new colour (that is, one that is not already installed in the colour map), MS-Windows either assigns it to a spare entry, or allocates the **closest match** if the colour map is full. The value of *Bits* and CMap after *Dyalog APL Language*: *Wc* reflect the actual colours allocated and may bear little resemblance to the values you assigned to these properties initially.

Note that if you are running 16 colours, MS-Windows reserves all 16 entries in the colour map for its own use. This means that on a 16-colour system, you **cannot** use any colours other than the default ones reserved by MS-Windows. In practice, the "standard" 16-colour CMap is shown in the following table.

Table: The default 16-colour CMap { #default-16-colour-cmap }

Bits[]	CMap			Colour
1	0	0	0	Black
2	128	0	0	Dark red
3	0	128	0	Dark Green
4	128	128	0	Olive Green
5	0	0	128	Dark Blue
6	128	0	128	Dark Magenta
7	0	128	128	Dark Cyan
8	128	128	128	Dark Grey
9	192	192	192	Light Grey
10	255	0	0	Red
11	0	255	0	Green
12	255	255	0	Yellow
13	0	0	255	Blue
14	255	0	255	Magenta
15	0	255	255	Cyan
16	255	255	255	White

If you are using a 256-colour set-up, the first 9 and the last 7 entries of the 256-colour CMap are the same as the first 9 and last 7 entries of the 16-colour CMap shown

above. The intervening entries represent additional colours or are initially unused (0 0 0). New colours that you specify will be allocated to unused entries until the table is full.

3.32 CalendarCols Property

Applies To: *Calendar*, *DateTimePicker*

Description

The *CalendarCols* property specifies the colours used for various elements in the *Calendar* object.

CalendarCols is a 6-element integer vector whose elements specify the colours as follows:

[1]	Background colour displayed between months
[2]	Background colour displayed within the month.
[3]	Text colour within a month
[4]	Background colour displayed in the calendar's title
[5]	Colour used to display text within the calendar's title
[6]	Colour used to display header day and trailing day text. Header and trailing days are the days from the previous and following months that appear on the current month calendar.

Each element of *CalendarCols* may be 0 (which means default colour), a negative singleton that specifies a particular Windows colour, or a 3-element integer vector of RGB values.

Note: At the time of writing, setting the first element of *CalendarCols* has no effect. Dyalog believes this to be a Windows problem that may be corrected in due course.

3.33 Cancel Property

Applies To: *Button*

Description

This property determines which (if any) Push button in a *Form* or *SubForm* is to be associated with the Escape key. It has the value 1 or 0.

Pressing the Escape key will generate a *Select* event on the *Button* whose *Cancel* property is 1, regardless of which object has the keyboard focus.

As only one button in a *Form* or *SubForm* can be the Cancel button, setting *Cancel* to 1 for a particular button automatically sets *Cancel* to 0 for all others in the same *Form*.

3.34 Caption Property

Applies To

Description

The *Caption* property is a character vector specifying fixed text associated with the object. For example, *Caption* defines the label on a *Button*, the title of a *Form*, *SubForm* or *MsgBox*, the heading in a *Group*, and the text of a *Menu* or a *MenuItem*.

For the *Root* object, *Caption* specifies the text displayed when Alt+Tab is used to switch to the Dyalog APL/W application. It may be used in conjunction with the *IconObj* property which specifies the name of an *Icon* object to be displayed alongside this text.

Its default value is an empty vector.

For a *Button* or *Label*, if the *Caption* property contains one or more linefeed characters (`␣`UCS 10) the text is top-left justified or, for a *Button* with *Style* 'Push', centre-justified ; and automatically wraps on white-space characters (such as space and tab) to fit in the width provided.

For controls that support this feature, a single ampersand (&) is used to designate that the following character (if present) is an access key or an accelerator key and that character is underlined. The ampersand is not itself displayed. To negate this feature and cause an ampersand to be displayed, it is necessary to specify "&&".

3.35 CaseSensitive Property

Applies To: *ComboEx*

Description

Specifies whether or not string searches in the items displayed by a ComboEx object will be case sensitive. Searching occurs when text is being typed into the edit box portion of the ComboEx

3.36 CellFonts Property

Applies To: *Grid*

Description

This property specifies the font objects to be used to display the *Values* in a *Grid* object.

CellFonts is either a single ref or a simple character vector, or a vector of refs or character vectors. If it specifies a single font object this will be used to draw the text in *all* of the cells in the *Grid*. If it specifies more than one font object, these are mapped to individual cells through the *CellTypes* property.

3.37 CellHeights Property

Applies To: *Grid*

Description

This property specifies the height of each row in a *Grid* object in the units specified by its *Coord* property. It may be a scalar or a vector whose length is the same as the number of rows implied by the *Values* property. If it is a scalar, it specifies a constant row height. If it is a vector it specifies the height of each row individually.

3.38 CellSelect Property

Applies To: *Grid*

Description

The *Grid* supports the selection of a contiguous block of cells by the user, using the mouse and/or the keyboard. The ability to select a range of cells is determined by the *CellSelect* property.

CellSelect may be a character vector or a vector of character vectors comprising the following:

'Rows'	User may select an entire row by clicking on a row title and may select multiple rows by dragging the mouse over contiguous row titles.
'MultiRows'	Same as 'Rows' , but user may additionally select several non-contiguous rows and blocks of rows using the Ctrl key.
'Columns'	User may select an entire column by clicking on a column title and may select multiple columns by dragging the mouse over contiguous column titles.
'MultiColumns'	Same as 'Columns' , but user may additionally select several non-contiguous columns and blocks of columns using the Ctrl key.
'Partial'	User may select any rectangular block of cells by either dragging the mouse or using Shift+cursor keys.
'MultiPartial'	Same as 'Partial' , but user may additionally select multiple rectangular blocks of cells using the Ctrl key.
'Whole'	User may select the entire <i>Grid</i> by clicking in the space to the left of the column titles and above the row titles.
'Any'	Same as ('Rows' 'Columns' 'Partial' 'Whole'). This is the default.
'Multi'	Same as ('MultiRows' 'MultiColumns' 'MultiPartial' 'Whole').
'None'	User may not select any cells in the <i>Grid</i> .

For example, the following expression would allow the user to select only whole rows and columns:

```
gridname WS 'CellSelect' ('Rows' 'Columns')
```

Setting CellSelect to ('Rows' 'Columns' 'Whole' 'Partial') is equivalent to setting it to 'Any'.

When the user performs a selection, the *Grid* generates a *GridSelect* event.

The range of cells currently selected is given by the *SellItems* property. You can obtain the current selection by querying this property with *Dyalog APL Language: Wg* and you can set it with *Dyalog APL Language: Ws*.

Note that the user may delete the contents of the selected range, or cut and copy them to the clipboard by pressing Delete, Shift+Delete or Ctrl+Insert respectively. The user may also replace the current selection with the contents of the clipboard by pressing Shift+Insert. These operations generate *GridDelete*, *GridCut*, *GridCopy* and *GridPaste* events which you may disable (by setting the event action code to -1 or to which you may attach a callback function).

If more than one block of cells has been selected, these operations are honoured only if the blocks begin and end on the same rows or begin and end on the same columns. If so, the data placed in the clipboard is the result of joining the blocks horizontally or vertically as appropriate.

You can also invoke these events as methods. This allows you to attach these actions to MenuItems and Buttons. For example, the following expression could be used to implement Cut as a MenuItem:

```
name WC 'MenuItem' 'Cu&t'('Event' 'Select' '#gridname.GridCut')
```

In addition to the ability to copy blocks of cells through the clipboard, the user may also drag a block of cells from one part of the *Grid* to another.

When the user places the mouse pointer over any of the four edges of a selected block of cells, the cursor changes from a cross to an arrow pointer. The user may now drag the border of the selected block to a new location. If the Ctrl key is pressed at the same time, the contents of the selected cells are *copied* to the new location. If not, the operation is a *move* and the original block of cells is cleared (emptied). In either case, the contents of the original block replace the contents of the target block (marked by the dragging rectangle) and the target block becomes selected. You may only move or copy a *single* block of cells in this way.

These operations generate a *GridDropSel* event. You may prevent the user from moving and copying blocks of cells by disabling this event (by setting its event action code to -1) or you may control these operations selectively with a callback function. Note that although the operation of *inserting* cells (using Ctrl+Shift) has not been implemented, you may provide this facility yourself with the information provided by the event message.

You may also move or copy a block of cells (which need not necessarily be selected) under program control by calling *GridDropSel* event as a method.

3.39 CellSet Property

Applies To: *Grid*

Description

This property identifies which cells in a *Grid* are *set* (that is, have values) and which are empty. Its purpose is to allow large numeric matrices containing blank cells to be displayed and edited efficiently.

The *CellSet* property is a Boolean matrix with the same shape as the *Values* property. If an element of *CellSet* is 0, the cell is defined to be empty. Empty cells are displayed as blank and the cell contents by the *Values* property are ignored.

A more direct way to handle empty cells is to set the corresponding elements of *Values* to empty vectors. However, if *Values* is otherwise entirely numeric, this makes the array nested when it would otherwise be simple. For large numeric matrices, this penalty can be severe. For example, a 100x100 array of 2-byte integer values occupies about 20Kb of workspace. Setting one or more elements of the array to an empty vector increases its size to 200Kb. However, because it is Boolean, the size of the *CellSet* property for a 100x100 array is only 1.27Kb and represents a significant space saving.

Note that if the *Values* property contains text and is therefore nested anyway, the *CellSet* property is not helpful in conserving workspace, although it may still be useful to separate empty cells from real data.

You can dynamically change a single element of *CellSet* using the *SetCellSet* method.

3.40 CellTypes Property

Applies To: *Grid*

Description

This property specifies the type of each cell in a *Grid* object. It is a matrix whose elements are origin-1 indices into other property arrays (*FCol*, *BCol*, *CellFonts* and *Input*).

For example, if *CellTypes*[1;1] is 3, the first cell in the *Grid* is displayed using the foreground colour specified by the 3rd element of *FCol*, the background colour specified by the 3rd element of *BCol*, and so forth. Note however that scalar property arrays are extended if necessary. Therefore if you require 5 different foreground colours but only one background colour, *BCol* need specify only a single colour.

You can dynamically change a single element of *CellTypes* using the *SetCellType* method.

3.41 CellWidths Property

Applies To: *Grid*

Description

This property specifies the width of each column in a *Grid* object in the units specified by its *Coord* property. It may be a scalar or a vector whose length is the same as the number of columns implied by the *Values* property. If it is a scalar, it specifies a constant column width. If it is a vector it specifies the width of each column individually.

3.42 Changed Property

Applies To: *ButtonEdit, Edit, PropertyPage, RichEdit, Spinner*

Description

The *Changed* property, in conjunction with the *Change* event, provides the means to control the validation of an object after the user has finished interacting with it.

Initially, the value of the *Changed* property of an object is set to 0. When the user gives the focus to the object and causes either the *Text* or (in the case of a *Spinner*) the *Thumb* property to be altered, the *Changed* property is immediately set to 1. When the object loses the input focus and the value of the *Changed* property is 1, the object generates a *Change* event. The value of the *Changed* property is then determined as follows:

If there is no callback function attached to the *Change* event, or if the *Change* event is disabled, the *Changed* property is reset to 0.

If an attached callback returns no result or returns 1, the *Change* property is reset to 0.

If an attached callback function returns 0, the *Changed* property is not altered and remains set to 1. The object will therefore generate another *Change* event when the user next tries to leave it, even if the text and/or *Thumb* are not altered this time.

Note that the object generates a *Change* event when it loses the focus *only* if the value of the *Changed* property is 1 at the time.

3.43 CharFormat Property

Applies To: *RichEdit*

Description

The CharFormat property describes or applies formatting to the currently selected text in a *RichEdit* object. If the selection is empty, it reports or specifies the default character formatting for the object. It is a 5-element nested array structured as follows:

[1]	A vector of character vectors which describes the text attributes and is comprised of the following keywords: 'Autocolour' default colour (Windows text colour) 'Bold' bold text 'Italic' bold text 'Underline' underlined 'StrikeOut' line through text 'Protected' protected (read-only) text
[2]	A character vector that specifies the face name of the font used to draw the text
[3]	Character height in <i>Twips</i> .
[4]	Text colour. A single integer or an enclosed vector of 3 RGB values. The default is 0 which implies the standard Windows text colour.
[5]	Integer specifying the vertical offset of the character from the base line in <i>Twips</i> . This is used to specify superscript (positive offset) and subscript (negative offset) symbols. The default value is 0.

When you set the character format using `WC` or `WS` the following rules apply:

If you just want to set a single text attribute (element 1) you may specify a simple vector, for example (`WS 'CharFormat' 'Protected'`) is valid and will add the protected text attribute to the current set of text attributes.

To unset a text attribute (element 1) you must insert the tilde (~) character before the name of the attribute. For example, the expression (`WS 'CharFormat' '~Bold'`) will turn the bold text attribute off.

You need only specify the number of elements required, but you must insert proper values for the elements you wish to remain unaltered. However, you may use '' in the first element to leave the text attributes unchanged.

If there is no text selected, CharFormat specifies the *default* character format, that is, the format that will be used to draw the next (and subsequent) characters that the user enters. If there *is* text selected it specifies the format of the selected block of text. If the format is not strictly homogeneous, ¶WG may report the format of the first character in the selected block, or, if the block contains characters which use completely different fonts, the result of (¶WG 'CharFormat') will be empty.

(¶WS 'CharFormat' ...) will set the format of the currently selected block of text. To set the format of an arbitrary block of text you must select it first using (¶WS 'SelText' ...).

3.44 CharSet Property

Applies To: *Font*

Description

CharSet is an integer that specifies the character encoding of the *Font* object.

This property applies to the Classic Edition only. In the Unicode Edition, its value typically has no effect.

The following table illustrates some of the character set encodings supported by Windows. Note that this set may vary according to the edition of Windows that is installed.

Language	CharSet
Western (ANSI)	0
Hebrew	177
Arabic	178
Greek	161
Turkish	162
Baltic	186
Central European	238
Cyrillic	204
Vietnamese	163

Windows fonts typically contain glyphs for the ASCII character set in their first 128 positions, and glyphs for the Western European character set in positions 129-256. Additional sets of character glyphs are stored in positions 257 onwards in what are sometimes referred to as codepages.

When you change the character set encoding, to (say) Greek (161), the set of Greek characters are mapped into the top 128 positions of the font.

For example, if the CharSet is 0 (ANSI), the character code Hex EC is displayed as ì (i-grave). However, if you change CharSet to 161 (Greek), the same character code is displayed as the Greek μ.

The following example illustrates how the character string 'ôï üíîá ïïö Ýéíáé Ðåôå' appears differently according to the value of CharSet.



```

'F'WC'FORM' 'CharSet Property'('Size' 300 400)('Coord' 'Pixel')
F.Coord←'Prop'

'F.F1'WC'Font' ('Size' 32) ('CharSet' 0)
'F.F2'WC'Font' ('Size' 32) ('CharSet' 161)
'F.F3'WC'Font' ('Size' 32) ('CharSet' 178)

STRING←'ôï üíîá ïïö Ýéíáé Đăôă'

'F.E1'WC'Edit'STRING(10 10)(0 80)('Font' 'F.F1')
'F.E2'WC'Edit'STRING(40 10)(0 80)('Font' 'F.F2')('Translate' 'ANS
I')
'F.E3'WC'Edit'STRING(70 10)(0 80)('Font' 'F.F3')('Translate' 'ANS
I')

```

Note

Setting Translate to 'ANSI' means that characters entered into the Edit fields using a corresponding National Language keyboard (Greek and Arabic) will appear in that language as intended.

3.45 CheckBoxes Property

Applies To: *ListView*, *TreeView*

Description

The CheckBoxes property specifies whether or not check boxes are displayed alongside items in a *ListView* or *TreeView* object.

CheckBoxes is a single number with the value 0 (check boxes are not displayed) or 1 (check boxes are displayed); the default is 0.

For a *TreeView*, CheckBoxes will only be honoured if the items have pictures associated with them (via the *ImageListObj* and *ImageIndex* properties).

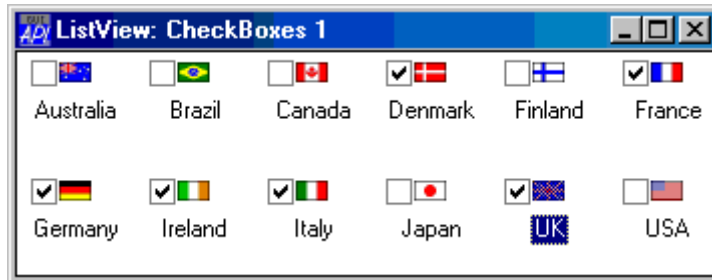
For a *ListView*, CheckBoxes applies to all settings of the *View* property.

The `GetItemState` method can be used to determine if a specific item in a *ListView* or *TreeView* is checked. The result of the method will have the 13th bit set if the item is checked.

```
STATE ← Form.ListView.GetItemState 11
13 > φ(32ρ2) ∓ STATE
```

1

The picture below illustrates the effect on the appearance of a *ListView* object, of setting `CheckBoxes` to 1.



3.46 Checked Property

Applies To: *MenuItem*

Description

This property determines whether or not a check mark is displayed alongside the text in a *MenuItem*. It is a single number with the value 0 (not checked) or 1 (checked). The default is 0.

3.47 ChildEdge Property

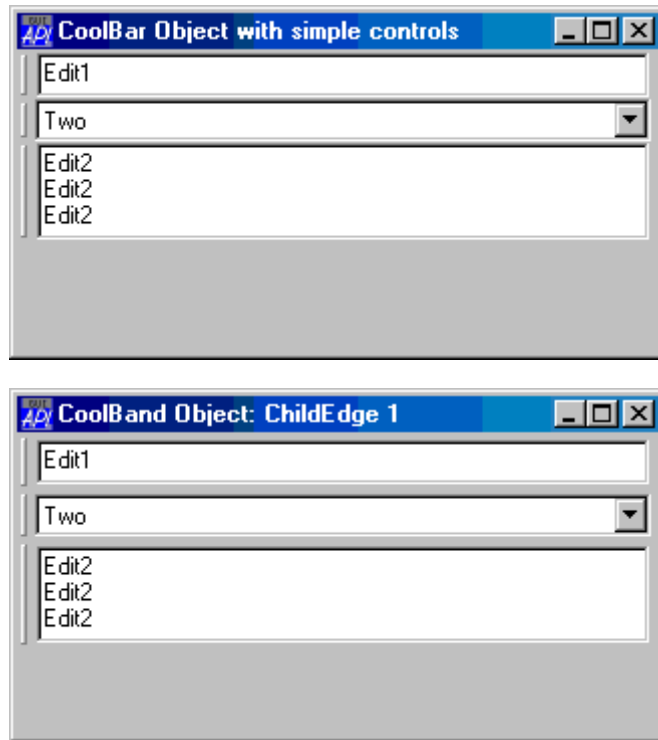
Applies To: *CoolBand*

Description

The `ChildEdge` property specifies whether or not the *CoolBand* leaves space above and below its child window.

`ChildEdge` is a single number with the value 0 (no space) or 1 (space is provided); the default is 0.

The following pictures illustrate the effect of the `ChildEdge` property.



3.48 ChildList Property

Applies To

Description

This is a read-only property that reports the names of those objects that may be created as children of the object in question. It is a vector of character vectors in which the order of the items is not significant.

3.49 CircleToday Property

Applies To: *Calendar*, *DateTimePicker*

Description

The *CircleToday* property specifies whether or not a circle is drawn around the *Today* date in a *Calendar* object or in the drop down calendar in a *DateTimePicker*, when the month containing that date is visible.

CircleToday is a single number with the value 0 (a circle is *not* drawn) or 1 (a circle is drawn); the default is 1.

See also *HasToday* property.

3.50 ClassID Property

Applies To: *ActiveXControl*, *OCXClass*, *OLEClient*, *OLEServer*

Description

The ClassID property specifies the class identifier (usually abbreviated to CLSID) of an APL object that is used to represent a COM object. The CLSID is a globally unique identifier (*GUID*) that uniquely identifies the object.

When you create or recreate an *ActiveXControl* or *OLEServer* using `□WC`, you may specify ClassID. This allows you to re-use a value that was previously allocated to that control by the system. However, you should not specify any other value because that value could be allocated now or in the future to another object *on any other computer in the world*. Otherwise, a new ClassID is automatically allocated by the system.

Note that the CLSID is not actually recorded on your computer (in the registry) until you register it using `)SAVE` or *Make OCX*, or by executing the *OLERegister* method.

3.51 ClassName Property

Applies To: *ActiveXControl*, *OCXClass*, *OLEClient*, *OLEServer*

Description

For an *OLEClient*, the ClassName property specifies the name of the OLE object to which an *OLEClient* object named by the left argument of `□WC` is to be connected. Similarly, for a *NetControl* the ClassName property specifies the name of the .NET class to be instantiated. Note that ClassName is mandatory for `□WC` and may not subsequently be changed using `□WS`.

For an *ActiveXControl* or *OLEServer*, ClassName specifies the external name with which the object is registered, and by which it is referenced by other applications.

For an *ActiveXControl*, the external name is "Dyalog xxx Control, where xxx is the value of the ClassName property, or, if ClassName is not specified, the name of the *ActiveXControl* namespace.

For an *OLEServer*, the external name is "Dyalog.xxx" where xxx is derived in the same way.

For a *NetControl*, the external name is the name of the .NET class which must be expressed relative to a corresponding element of `USING`. For example, to load one of the standard .NET controls:

```
USING, < 'System.Windows.Forms, system.windows.forms.dll'
```

3.52 ClipCells Property

Applies To: *Grid*

Description

This property determines whether or not the *Grid* displays partial cells. The default is 1. If you set *ClipCells* to 0, the *Grid* displays only complete cells and automatically fills the space between the last visible cell and the edge of the *Grid* with the *GridBCol* colour.

The first picture below shows a default *Grid* (*ClipCells* is 1) in which the third column of data is in fact incomplete (clipped), although this is by no means apparent to the user. The second picture shows the effect on the *Grid* of setting *ClipCells* to 0 which prevents such potential confusion.

ClipCells is 1 (the default)			
	A	B	C
1	13	75	4
2	51	83	
3	68	58	9
4	76	26	
5	98	72	7

ClipCells is 0			
	A	B	
1	13	75	
2	51	83	
3	68	58	
4	76	26	
5	98	72	

3.53 ColLineTypes Property

Applies To: *Grid*

Description

This property specifies the appearance of the vertical grid lines in a *Grid* object.

ColLineTypes is an integer vector, whose length is normally equal to the number of columns in the *Grid*. Each element in ColLineTypes specifies an index into the *GridLineFCol* and *GridLineWidth* properties, thus selecting the colour and width of the vertical grid lines.

For example, if ColLineTypes[1] is 3, the first vertical grid line in the *Grid* is displayed using the colour specified by the 3rd element of *GridLineFCol*, and the width specified by the 3rd element of *GridLineWidth*.

Note that ColLineTypes is not $\square 10$ dependent, and the value 0 is treated the same as the value 1; both selecting the *first* colour and line width specified by *GridLineFCol* and *GridLineWidth* respectively.

The default value of ColLineTypes is an empty numeric vector ($\emptyset$). If so, all vertical grid lines are drawn using the first element of *GridLineFCol* and *GridLineWidth*.

A vertical grid line is drawn down the right edge of its associated column. One pixel is drawn *inside* the column of cells; additional pixels (if any) are drawn *between* that column of cells and the next one to its right.

3.54 ColSortImages Property

Applies To: *Grid*

Description

The ColSortImages property identifies the names of, or refs to, up to three *Bitmap* objects that are used to specify the sort images for a *Grid* object.

If ColSortImages is not specified, default images are used.

The *Bitmap* specified by the 1st element of ColSortImages is used to display columns that are sorted down.

The *Bitmap* specified by the 2nd element of ColSortImages is used to display columns that are unsorted.

The *Bitmap* specified by the 3rd element of ColSortImages is used to display columns that are sorted up.

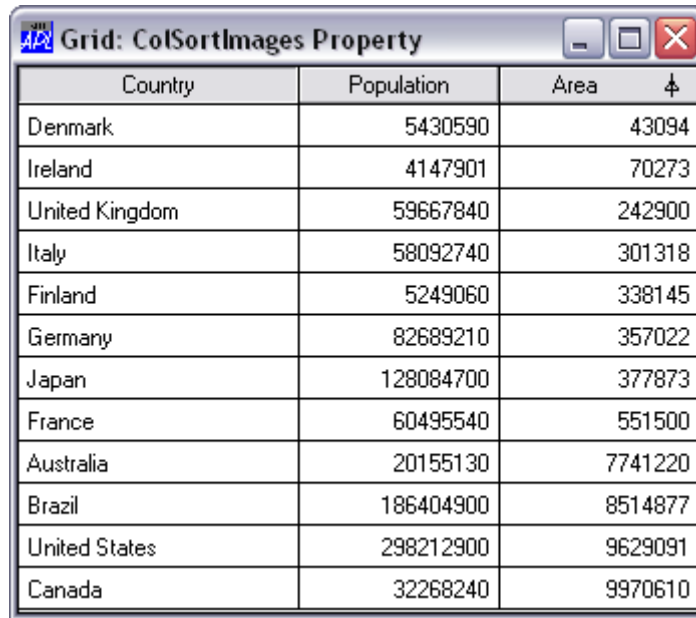
```
'F'WC'Form' 'Grid: ColSortImages Property'
F.(Coord Size)←'Pixel'(313 341)
'F.fnt'WC'Font' 'APL385 Unicode' 16
F.FontObj←F.fnt
BK←16 16p256LWhite←255 255 255

'F.gu'WC'Bitmap'('CBits'BK)('MaskCol'White)
'F.gu.'WC'Text' '▲'(0 3)
'F.gd'WC'Bitmap'('CBits'BK)('MaskCol'White)
'F.gd.'WC'Text' '▼'(0 3)

'F.G'WC'Grid'('Posn' 0 0)F.Size
F.G.Values←#.(COUNTRIES,POPULATION,[1.5]AREA)
F.G.ColTitles←'Country' 'Population' 'Area'
F.G.CellWidths←140 100 100
F.G.TitleWidth←0

F.G.ColSortImages←'F.gd' '' 'F.gu'
F.G.(Values←Values[▲Values[;3];])

F.G.ColSorted 3 1
```

Country	Population	Area
Denmark	5430590	43094
Ireland	4147901	70273
United Kingdom	59667840	242900
Italy	58092740	301318
Finland	5249060	338145
Germany	82689210	357022
Japan	128084700	377873
France	60495540	551500
Australia	20155130	7741220
Brazil	186404900	8514877
United States	298212900	9629091
Canada	32268240	9970610

3.55 ColTitle3D Property

Applies To: *ListView*

Description

The ColTitle3D property is a Boolean value that specifies whether or not the column titles in a *ListView* object are displayed with a 3-dimensional effect. Its default value is 1. A column heading with a 3-dimensional button appearance may be used to imply that the user may click on it to sort by the values in that column.

ColTitle3D is only relevant if *View* is 'Report' and *Header* is 1. Note that this property may only be set by `WC` and may not subsequently be changed using `WS`.

3.56 ColTitleAlign Property

Applies To: *Grid, ListView*

Description

The ColTitleAlign property specifies the alignment of column titles. For a *ListView* object this is only relevant only when the *View* property is set to 'Report'.

ColTitleAlign is either a simple character vector, or a vector of character vectors with one element per column.

For a *Grid*, ColTitleAlign may be: 'Top', 'Bottom', 'Left', 'Right', 'Centre', 'TopLeft', 'TopRight', 'BottomLeft', or 'BottomRight'.

For a *ListView* object, ColTitleAlign may be 'Left', 'Right' or 'Centre'. Also, for a *ListView* the column data itself is aligned likewise. Note that the *first* column in a *ListView* is always left-aligned regardless of the setting of ColTitleAlign. This is a Windows restriction.

Note that both spellings 'Centre' and 'Center' are accepted.

3.57 ColTitleBCol Property

Applies To: *Grid*

Description

The ColTitleBCol property specifies the background colour of the column titles in a *Grid* object

ColTitleBCol may be a scalar that specifies a single background colour to be used for all of the column titles, or a vector that specifies the background colour of each of the column titles individually. An element of ColTitleBCol may be an enclosed 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or it may be a scalar that defines a standard Windows colour element (see *BCol* for details). Its default value is 0 which obtains the colour defined for Button Face.

3.58 ColTitleDepth Property

Applies To: *Grid*

Description

ColTitleDepth specifies the structure of a set of hierarchical column titles. It is an integer vector with the same length as the *ColTitles* property. A value of 0 indicates that the corresponding element of *ColTitles* is a top-level title. A value of 1 indicates that the corresponding title is a sub-title of the most recent title whose ColTitleDepth is 0; a value of 2 indicates that the corresponding title is a sub-title of the most recent title whose ColTitleDepth is 1, and so forth. For example:

```

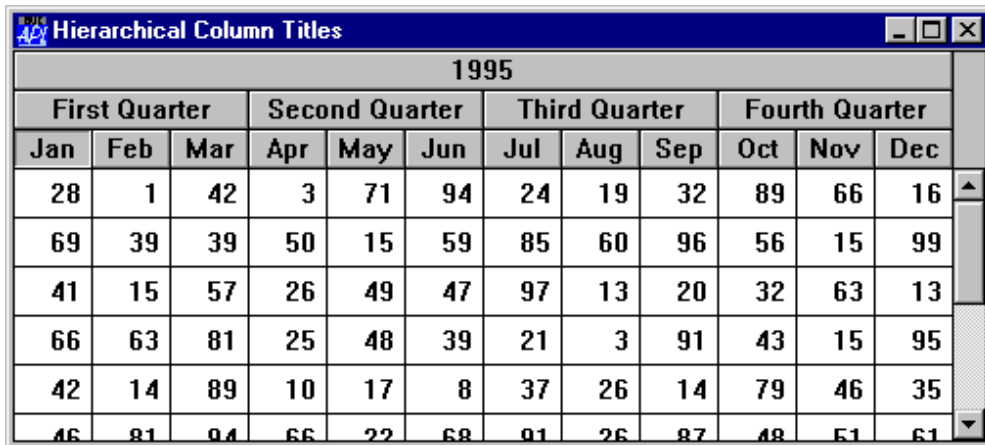
'F'□WC'Form'('Coord' 'Pixel')('Size' 200 498)
'F'□WS'Caption' 'Hierarchical Column Titles'
'F.G'□WC'Grid'(?10 12p100)(0 0)(200 498)
'F.G'□WS('TitleWidth' 0)('TitleHeight' 60)
'F.G'□WS'CellWidths' 40

Q1←'First Quarter' 'Jan' 'Feb' 'Mar'
Q2←'Second Quarter' 'Apr' 'May' 'Jun'
Q3←'Third Quarter' 'Jul' 'Aug' 'Sep'
Q4←'Fourth Quarter' 'Oct' 'Nov' 'Dec'

CT←(c'1995'),Q1,Q2,Q3,Q4
CD←0,16p1 2 2 2

'F.G'□WS('ColTitles'CT)('ColTitleDepth'CD)

```



1995											
First Quarter			Second Quarter			Third Quarter			Fourth Quarter		
Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
28	1	42	3	71	94	24	19	32	89	66	16
69	39	39	50	15	59	85	60	96	56	15	99
41	15	57	26	49	47	97	13	20	32	63	13
66	63	81	25	48	39	21	3	91	43	15	95
42	14	89	10	17	8	37	26	14	79	46	35
46	81	94	66	22	68	91	26	87	48	51	61

Note that the *LockColumns* method is not supported in combination with hierarchical column titles.

3.59 ColTitleFCol Property

Applies To: *Grid*

Description

The *ColTitleFCol* property specifies the colour of the column titles in a *Grid* object

ColTitleFCol may be a scalar that specifies a single colour to be used for all of the column titles, or a vector that specifies the colour of each of the column titles individually. An element of *ColTitleFCol* may be an enclosed 3-element vector of integer

values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or it may be a scalar that defines a standard Windows colour element (see *BCol* for details). Its default value is 0 which obtains the colour defined for Button text.

3.60 ColTitles Property

Applies To: *Grid, ListView*

Description

This property specifies the headings that are displayed above the columns in a *Grid* object. If specified, it must be a vector of character vectors or matrices whose length is the same as the number of columns implied by the *Values* property. The default value of *ColTitles* is an empty character vector. In this case, the system displays "standard" spreadsheet column titles A-Z, AA-AZ, BA-BZ and so forth.

To disable the display of column titles in a *Grid*, you should set the *TitleHeight* property to 0.

3.61 Collate Property

Applies To: *Printer*

Description

Specifies whether or not multiple copies of printer output are collated.

Collate is a single number with the value 0 or 1. If Collate is 1, multiple copies of output are collated separately. If Collate is 0, copies are uncollated on output.

Collate is ignored unless *Copies* is >1.

The default value for Collate is derived from the current printer setting and Collate is only effective if the printer supports this capability.

3.62 ColorMode Property

Applies To: *Printer*

Description

Specifies whether or not printing is done in colour.

ColorMode is a single number with the value 0 or 1. If ColorMode is 1, printing is done in colour. If ColorMode is 0, printing is done using black ink only.

This property only applies to colour printers.

3.63 ColumnWidth Property

Applies To: *List*

Description

This property specifies the column width in pixels of a multi-column *List* object. See *MultiColumn* property for details.

3.64 Container Property

Applies To: *ActiveXControl*

Description

The Container property is a read-only property whose value is the `IOleObject` of an *ActiveXContainer* object that represents the *ActiveX Site* object of the application that is hosting the *ActiveXControl*.

The value of Container may be converted to a namespace using `IOleObject::GetNamespace` or `IOleObject::GetContainer`.

The resulting object may then be used to obtain the values of ambient properties, or to access methods exposed by the host application via OLE interfaces. See *OLEQueryInterface*.

3.65 Coord Property

Applies To

Description

This property defines an object's co-ordinate system. It is a character string with one of the following values; 'Inherit', 'Prop', 'Pixel', 'RealPixel', 'ScaledPixel', 'User' or 'Cell' (graphics children of a *Grid* only).

If `Coord` is `'Inherit'`, the co-ordinate system for the object is **inherited** from its parent. Note that the default value of `Coord` for the system object `'.'` is `'Prop'`, so by default all objects created by *Dyalog APL Language: Wc* inherit `'Prop'`.

If `Coord` is `'Prop'`, the origin of the object's parent is deemed to be at its top left interior corner, and the scale along its x- and y-axes is 100. The object's position and size (`Posn` and `Size` properties) are therefore specified and reported as a percentage of the dimensions of the parent object, or, for a *Form*, of the screen.

If `Coord` is `'RealPixel'`, the origin of the object's parent is deemed to be at its top left interior corner, and the scale along its x- and y-axes is measured in physical pixel units. The object's position and size (`Posn` and `Size` properties) are therefore reported and set in physical pixel units. If you set `Coord` on the system object to `'Pixel'`, the value of its `Size` property gives you the resolution of your screen. Note that pixels are numbered from 0 to (`Size - 1`).

If `Coord` is `'ScaledPixel'` the number of pixels specified for `Posn`, `Size`, and other such properties will be automatically scaled by Dyalog APL according to the user's chosen display scaling factor. So if you specify an Edit object to be 80 pixels wide and 20 pixels high, and the user's scaling factor is 150%, Dyalog will automatically draw it 120 pixels wide and 30 pixels high. Dyalog will also de-scale coordinate values reported by `⌈WG` and event messages.

If `Coord` is `'Pixel'`, it is interpreted as either `'RealPixel'` or `'ScaledPixel'` according to the value of the **Dyalog_Pixel_Type** parameter, which is either `ScaledPixel` or `RealPixel`. See *Installation/Configuration: Dyalog Pixel Type*.

If this parameter is not specified, the default is `RealPixel`. So by default, when you set `Coord` to `Pixel`, it will be treated as `RealPixel`.

If `Coord` is `'User'`, the origin and scale of the co-ordinate system are defined by the values of the `YRange` and `XRange` properties **of the parent object**. Each of these is a 2-element numeric vector whose elements define the co-ordinates of top left and bottom right interior corners of the (parent) object respectively.

Note that if `Coord` is `'User'` and you change the values of `YRange` and/or `XRange` of the parent, the object (and all its siblings with `Coord 'User'`) are redrawn (and clipped) according to the new origin and scale defined for the parent. The values of their `Posn`, `Size` and `Points` properties are unaffected. Changing `YRange` and/or `XRange` therefore provides a convenient and efficient means to **"pan and zoom"**.

The `Coord` property for graphic objects created as children of a `Grid` may also be set to `Cell`. Apart from being easier to compute, a graphic drawn using cell coordinates will expand and contract when the grid rows and columns are resized.

EXAMPLE

This statement creates a button 10 pixels high, 20 pixels wide, and 5 pixels down and along from the top-left corner of the parent *Form T*.

```
'T.B1'□WC'Button' 'OK'(5 5)(10 20)('Coord' 'Pixel')
```

If you set *Coord* to 'RealPixel' in the *Root* object '.', then query its *Size*, you get the dimensions of the screen in pixels, that is:

```
'.' □WS 'Coord' 'RealPixel'
'.' □WG 'Size'
480 640
```

If you set *Coord* to 'ScaledPixel' in the *Root* object '.', then query its *Size*, you get the virtual resolution of the screen, that is:

```
'.' □WS 'Coord' 'ScaledPixel'
'.' □WG 'Size'
1080 1920
```

3.66 Copies Property

Applies To: *Printer*

Description

Specifies the number of copies to be printed.

Copies is a non-zero scalar integer value whose default is defined by the current printer settings.

3.67 CornerTitleBCol Property

Applies To: *Grid*

Description

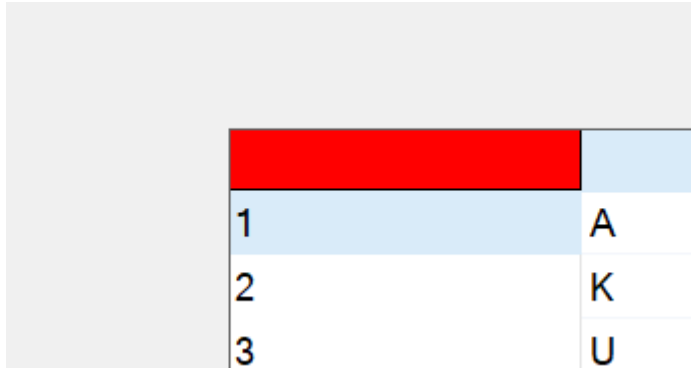
This property specifies the colour used to fill the area in the left corner a *Grid*. This is the rectangle above the row titles and to the left of the column titles.

CornerTitleBCol may be a 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or it may be a

scalar that defines a standard Windows colour element (see *BColor* for details). Its default value is 0 which means that the colour derives from your current Windows colour scheme.

EXAMPLE

```
f.g.CornerTitleBColor←255 0 0
```



3.68 Cue Property

Applies To: *ButtonEdit*, *Edit*

Description

This property specifies optional text to be displayed when a *ButtonEdit* or an *Edit* object is empty. For an *Edit* object it applies only if the *Style* of the *Edit* object is 'Single'.

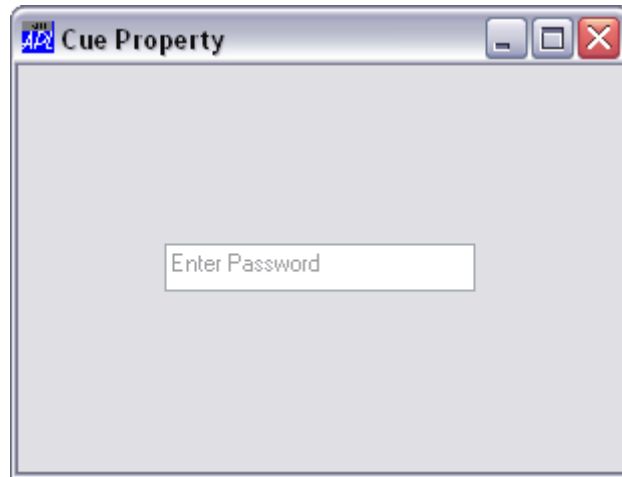
Note

This feature only applies if *Native Look and Feel* is enabled.

The Boolean property *ShowCueWhenFocused* determines whether or not the cue should also be displayed once the user has tabbed into or clicked on the input field (and thus given it the focus).

EXAMPLE

```
'F' WC 'Form' 'Cue Property'
'F.E' WC 'Edit'
F.E.Cue←'Enter Password'
```



3.69 CurCell Property

Applies To: *Grid*

Description

This property specifies or reports the current cell in a *Grid* object. The current cell is the one that is currently addressed by the user. The current cell is identified visibly if it has an associated object, other than an external. *Label* object.

3.70 CurrentColor Property

Applies To: *ColorButton*

Description

The *CurrentColor* property is a 3-element integer vector that specifies and reports the RGB value of the currently selected colour in a *ColorButton* object. Its default value is (0 0 0) which is black.

3.71 CurrentState Property

Applies To: *TCPSocket*

Description

The *CurrentState* property is a read-only property that reports the current state of a *TCPSocket* object. Its possible values and their means are as follows:

CurrentState	Description
'Open'	a client socket that is not yet connected or a UDP socket
'Bound'	a server socket that has been bound
'Listening'	a server socket to which a client has not yet connected
'Connected'	a client or server socket that is connected
'IHaveClosed'	a temporary state on the way to Closed
'PartnerHasClosed'	a temporary state on the way to Closed
'Closed'	a socket that has been closed by both client and server

3.72 CursorObj Property

Applies To

Description

This property is used to associate a particular cursor with an object.

Its value is either a simple scalar number which specifies a standard Windows cursor, or the name of, or ref to, a *Cursor* object. The standard Windows cursors are :

0	arrow (Windows default)
1	hourglass
2	crosshair
3	I-Beam
4	crossing vertical/horizontal double-headed arrows
5	diagonal double-headed arrows (left-to-right)
6	vertical double-headed arrows
7	diagonal double-headed arrows (right-to-left)
8	horizontal double-headed arrows
9	upward pointing arrow
10	box
11	crossing vertical/horizontal double-headed arrows
12	no-entry sign
13	arrow with hourglass
14	pointing hand

If `CursorObj` is set to anything other than an empty vector (which is the default) it defines the appearance of the cursor when the mouse pointer is moved into the object. If `CursorObj` is an empty vector, the shape of the cursor remains unchanged when the mouse pointer enters the object. In effect, the cursor is "inherited" from its parent. Exceptions to this rule are certain objects which have special cursors by default.

If the value of `CursorObj` for the *Root* object is set to anything other than an empty vector, it applies to **all** *Forms* and their children, irrespective of their own `CursorObj` values. Therefore, if you want to indicate that your application is "working" and is not responsive to input, you can simply do:

```
',' WS 'CursorObj' 1 A Hourglass cursor
```

Then to reset the application you do:

```
',' WS 'CursorObj' ''
```

3.73 CustomColors Property

Applies To: *ColorButton*

Description

The CustomColors property is a 1-row, 16-column nested matrix which specifies the RGB values of the colours displayed in the *Custom colors* section of the Windows colour selection dialog box when displayed by a *ColorButton* object.

Each element of CustomColors is a 3-element integer vector specifying an RGB colour value.

By default, each element of CustomColors is (0 0 0). If the user selects a new custom colour from the Windows colour selection dialog box, its value will be reported by CustomColors. CustomColors must always have shape (1 16).

Note that CustomColors is maintained separately for each separate *ColorButton*, and CustomColors defaults to (1 16) of (0 0 0) for each new *ColorButton* that you create. If you want to maintain a global custom colour table for your application, you must do this yourself.

3.74 CustomFormat Property

Applies To: *DateTimePicker*

Description

Specifies a custom format for the date/time display in a *DateTimePicker*.

CustomFormat is a character vector that may contain a mixture of date/time format elements and body text. The date/time elements are replaced by the actual date/time values when the object is displayed. The body text is displayed *as-is*. Note that CustomFormat may only be specified when the *DateTimePicker* object is created.

The date/time elements are defined by the following groups of characters, notice that they are case-sensitive:

Element	Description
d	The one- or two-digit day.
dd	The two-digit day. Single-digit day values are preceded by a zero.
ddd	The three-character weekday abbreviation.
dddd	The full weekday name.
h	The one- or two-digit hour in 12-hour format.
hh	The two-digit hour in 12-hour format. Single-digit values are preceded by a zero.
H	The one- or two-digit hour in 24-hour format.
HH	The two-digit hour in 24-hour format. Single-digit values are preceded by a zero.
m	The one- or two-digit minute.
mm	The two-digit minute. Single-digit values are preceded by a zero.
M	The one- or two-digit month number.
MM	The two-digit month number. Single-digit values are preceded by a zero.
MMM	The three-character month abbreviation.
MMMM	The full month name.
t	The one-letter AM/PM abbreviation (that is, AM is displayed as "A").
tt	The two-letter AM/PM abbreviation (that is, AM is displayed as "AM").
yy	The last two digits of the year (that is, 1996 would be displayed as "96").
yyyy	The full year (that is, 1996 would be displayed as "1996").

The body text is defined by sub-strings contained within single quotes. For example, to display the current date with the format "Today is: 04:22:31 Tuesday Mar 23, 1996", the format string is defined as follows:

```
CustomFormat
'Today is: 'hh':'m':'s dddd MMM dd', 'yyyy'
```

To include a single quote in your body text, use two consecutive single quotes. For example, to produce output that looks like: "Don't forget Mar 23, 1996", CustomFormat should be specified as follows:

```
CustomFormat  
'Don''t forget' MMM dd', ' yyyy
```

Note

Non-format characters that are not delimited by single quotes will result in unpredictable display by the *DateTimePicker* object.

3.75 Data Property

Applies To

Description

This property allows you to associate arbitrary data with an object. The value of the Data property may be any APL array.

3.76 DateTime Property

Applies To: *DateTimePicker*

Description

Specifies the value of date/time in a *DateTimePicker*.

The DateTime property represents the date and time value that is currently displayed in a *DateTimePicker* object.

It is normally a 4-element integer vector containing the date (as an IDN), hour, minutes and seconds respectively.

However, if the checkbox shown in the object is unset (see *HasCheckBox*), the value of DateTime will be 0 (zilde).

3.77 DbClickToggle Property

Applies To: *CoolBar*

Description

The *DbClickToggle* property specifies whether or not the user must single-click or double-click to toggle a child *CoolBand* between its maximised or minimised state.

DbClickToggle is a single number with the value 0 (single-click toggles state) or 1 (double-click toggles state); the default is 0.

3.78 Decimals Property

Applies To: *ButtonEdit*, *Edit*, *Label*, *Spinner*

Description

This property specifies the number of decimal places to which a number is to be displayed in an *Edit* or *Label* object with *FieldType* 'Numeric'. For an *Edit* object, *Decimals* also specifies the maximum number of digits that the user may enter after a decimal point.

The default value of *decimals* is -1 which allows any number of decimal places to be entered.

3.79 Default Property

Applies To: *Button*, *MsgBox*

Description

This property determines which of a set of push buttons in a *Form*, *SubForm* or *MsgBox* is the default button.

In a *Form* or *SubForm*, the *Default Button* will generate a *Select* event (30) when the user presses the Enter key, even though the *Default Button* may not have the focus at the time.

If however, the user explicitly shifts the focus to another *Push Button*, the automatic selection of the *Default Button* is disabled and the Enter key applies to the *Button* with the focus.

For a *Button*, the *Default* property has the value 1 or 0. As only one *Button* can be the *Default Button*, setting *Default* to 1 for a particular *Button* automatically sets *Default* to 0 for all others with the same parent.

In a *MsgBox*, *Default* specifies which button initially has the focus. It has the value 1, 2 or 3 corresponding to the three buttons that can be defined. See *Btns* property for further details.

3.80 DefaultColors Property

Applies To: *ColorButton*

Description

The *DefaultColors* property is a nested matrix which specifies the RGB values of the colours shown in the colour selection drop-down displayed by a *ColorButton* object.

The shape of *DefaultColors* determines the number of rows and columns in the colour selection drop-down.

Each element of *DefaultColors* is a 3-element integer vector specifying an RGB colour value.

3.81 Depth Property

Applies To: *TreeView*

Description

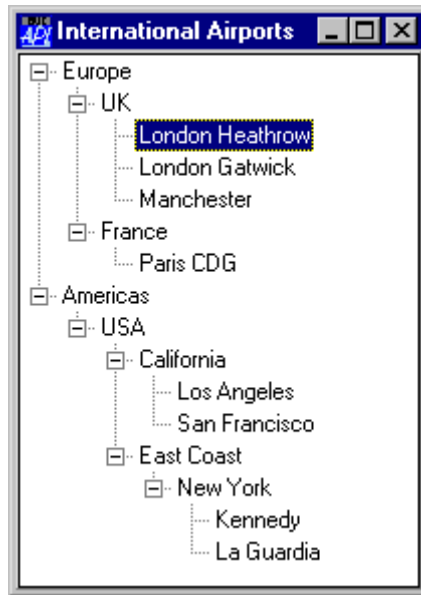
The *Depth* property specifies the structure of the items in a *TreeView* object. It is either a scalar 0 or an integer vector of the same length as the *Items* property.

A value of 0 indicates that the corresponding item is a top-level item. A value of 1 indicates that the corresponding item is a child of the most recent item whose *Depth* is 0; a value of 2 indicates that the corresponding item is a child of the most recent item whose *Depth* is 1, and so forth.

For example:

AIRPORTS	DEPTH	Description
Europe	0	Top-level (root) item
UK	1	1st sub-item of Europe
London Heathrow	2	1st sub-item of UK
London Gatwick	2	2nd sub-item of UK
Manchester	2	3rd sub-item of UK
France	1	2nd sub-item of Europe
Paris CDG	2	1st sub-item of France
Americas	0	Top-level (root) item
USA	1	1st sub-item of N.America
California	2	1st sub-item of USA
Los Angeles	3	1st sub-item of California
San Francisco	3	2nd sub-item of California
East Coast	2	2nd sub-item of USA
New York	3	1st sub-item of East Coast
Kennedy	4	1st sub-item of NY
La Guardia	4	2nd sub-item of NY

```
'F'WC'FORM' 'International Airports'
'F.TV'WC'TreeView'AIRPORTS(0 0)(100 100)('Depth'DEPTH)
```



3.82 DevCaps Property

Applies To: *Printer, Root*

Description

This property reports the device capabilities of the screen or printer. It is a 4-element nested vector as follows:

[1]	Height and Width:2-element numeric vector of device in pixels
[2]	Height and Width:2-element numeric vector of device in mm
[3]	Number of colours or $^{-1}$
[4]	Windows scaling factor as a percentage (100=no scaling). This value is the same as reported in the Display section of the Windows Control Panel

This property is useful if you want to make objects of a specific physical size. For example, to draw a 10mm square in a *Form* 'F' at (5,5):

```
Size ← 10× ⌈÷/2↑'. ' □WC 'DevCaps'
'F.R' □WC 'Rect' (5 5) Size ('Coord' 'Pixel')
```

Notes

- the physical size reported for the screen is typically only a *nominal* size, because, if you use a generic video driver, Windows has no way to tell what size of screen is attached to your computer.
- The number of colours is reported only if the device has a colour depth of no more than 8 bits per pixel. For devices with greater colour depths, -1 is returned.
- new elements may be added to DevCaps in future releases.

3.83 Directory Property

Applies To: *FileBox*

Description

The Directory property contains a simple character vector which specifies the initial directory from which a list of suitable files is displayed.

If, whilst interacting with the *FileBox*, the user changes directory and exits by pressing "OK" or by closing the *FileBox*, the value of the Directory property is updated accordingly.

3.84 Divider Property

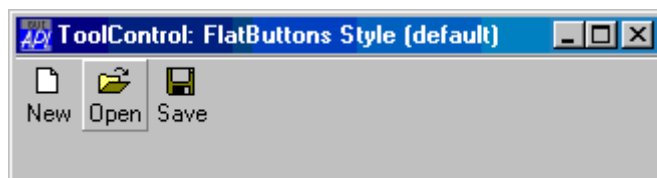
Applies To: *ToolControl*

Description

The Divider property controls the presence or absence of a recessed line drawn above, below, to the left of, or to the right of a *ToolControl* object.

Divider is a single number with the value 0 (dividing line is *not* drawn) or 1 (a dividing line *is* drawn); the default is 1.

The pictures below illustrates a *ToolControl* drawn with and without a divider.





3.85 DockChildren Property

Applies To: *CoolBar, Form, SubForm*

Description

The `DockChildren` property specifies the names of client objects that may be docked in a host object.

`DockChildren` may be a single ref or simple character scalar or vector, or a vector of refs or character vectors. Each item represents an object that may be docked. Notice that if you use a name, you must specify the simple name of the object, excluding any part of its full pathname that refers to a parent; that is, the specified names must not contain any leading pathname information.

If the name of, or ref to, a *dockable* object occurs in the `DockChildren` property, the host object will generate `DockMove` events when the client is dragged over it, and will generate a `DockAccept` event when a docking operation takes place.

If the name of, or ref to, the client object is not present in its `DockChildren` property, the object will not respond in any way as the client is dragged over it.

The following example shows the creation of 3 dockable forms, all of which are dockable in a host form called `h1`.

The first, `c1`, is a totally independent Form. When docked in `h1`, it will become a SubForm `h1 . c1`. When undocked, it will revert to an independent Form `c1`.

The second, `c2`, is created initially as a child of `h1` and will therefore be displayed above it in the window stacking order. When docked it will become a SubForm `h1 . c2`. When undocked, it will revert back to a dependent Form `h1 . c2`. In all cases, it appears on top of `h1`.

The third, `c3`, is created initially as a child of another Form, `h2`. When docked (in `h1`) it will become a SubForm `h1 . c3`. When undocked, it will become a dependent Form `h1 . c3`, and will therefore appear above `h1` in the stacking order.

```

'h1' WC 'Form' 'Host1'
'h2' WC 'Form' 'Host2'

'c1' WC 'Form' 'Client 1' ('Dockable' 'Always')
'h1.c2' WC 'Form' 'Client 2' ('Dockable' 'Always')
'h2.c3' WC 'Form' 'Client 3' ('Dockable' 'Always')

h1.DockChildren←'c1' 'c2' 'c3'

```

3.86 DockShowCaption Property

Applies To: *Form, SubForm*

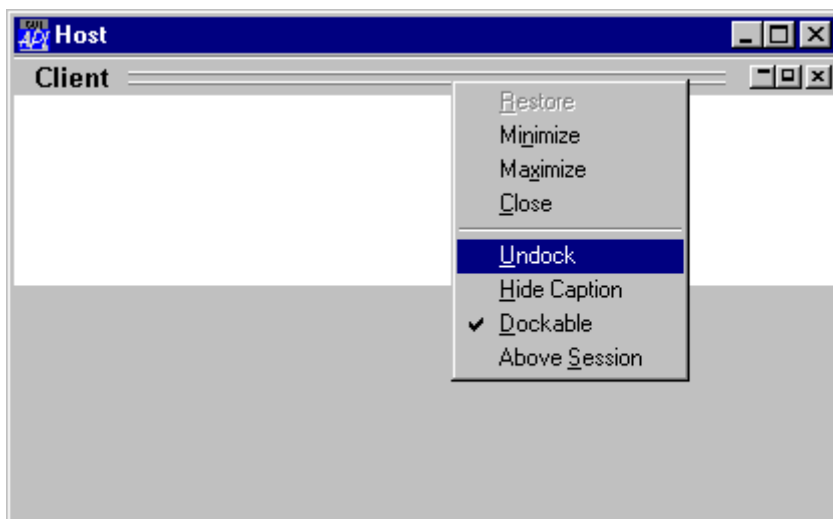
Description

The DockShowCaption property specifies whether or not a Form displays a title bar when it is docked as a SubForm.

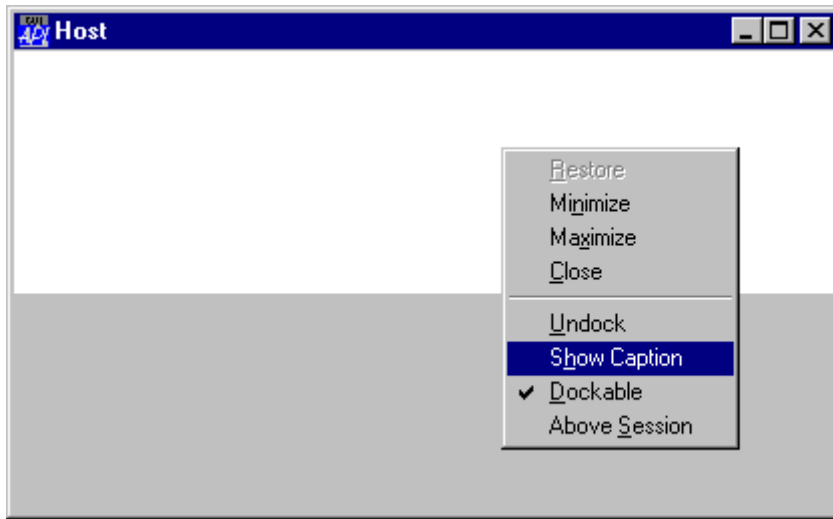
DockShowCaption is a single number with the value 0 or 1 (the default).

The DockShowCaption property may be toggled on and off by the user from the object's context menu.

The first picture below illustrates a Form, docked as a SubForm, whose DockShowCaption property is 1, but is about to be set to 0.



The next picture shows the same docked Form with DockShowCaption set to 0.



3.87 Dockable Property

Applies To: *CoolBand, Form, SubForm, ToolControl*

Description

The Dockable property specifies whether or not an object may be docked or undocked.

Dockable is a character vector containing 'Never' (the default), 'Always' or 'Disabled'.

If Dockable is 'Never', the object may not be docked or undocked by the user, and the docking menu items are not present in the object's context menu. This is the default.

If Dockable is 'Always', the object may be docked or undocked by the user, and the docking menu items are present in the object's context menu.

If Dockable is 'Disabled', the object may not currently be docked or undocked by the user, but the docking menu items are present in the object's context menu.

Note that by default, the user may switch between Dockable 'Always' and 'Disabled' by toggling the *Dockable* menu item. If you want to exercise full control over this property, you may implement your own context menu (see *ContextMenu Event*)

3.88 Docked Property

Applies To: *Form*, *SubForm*

Description

The Docked property is a read-only property that indicates whether or not an object is currently docked.

Docked is a single number with the value 0 (is not docked) or 1 (is docked).

3.89 DragItems Property

Applies To: *ListView*

Description

The DragItems property is Boolean and specifies whether or not the items in a *ListView* object may be drag/dropped by the user. Its default value is 1.

3.90 Draggable Property

Applies To

Description

This property determines whether or not an object may be the subject of a "drag and drop" operation.

Dragable is a single number with the value 0, 1 or 2. A value of 0 (which is the default) means that the object may not be drag/dropped. A value of 1 means that the object may be drag/dropped and that during the "drag" operation, a box representing the bounding rectangle around the object is displayed on the screen. A value of 2 means that the **outline** of the object is displayed as the object is dragged, or, if the object is an *Image* with a *Picture* property containing the name of an *Icon* object, the icon itself is displayed as it is dragged. For rectangular non-graphical objects, values of 1 and 2 are equivalent.

If an object is Draggable, the user may drag it by positioning the mouse pointer within the object, depressing the left mouse button, then moving the mouse with the button held down. During the drag operation, the mouse pointer is automatically changed to a "drag" symbol. The object is "dropped" by releasing the left mouse button. The effect

depends upon where it is dropped, and on the action associated with the *DragDrop* event for the object under the mouse pointer (if any).

If there is no object under the mouse pointer, the "drag and drop" operation is ignored. Otherwise, the object under the mouse pointer generates a *DragDrop* event.

If the object under the mouse pointer is the parent of the object that has been "dragged and dropped", the default action is for the system to move that object to the new location within its parent. If you wish to allow your user to freely move an object within its parent *Form* or *Group*, simply set its *Dragable* property to 1; the system will take care of the rest. If you want to allow the user to move an object, but you want to know about it when it happens, you can associate a callback function to the *DragDrop* event that queries the new position. To permit the operation to complete, the callback function should either not return a result or it should return something other than a scalar 0. To selectively disable movement, your callback function should return a scalar 0 in circumstances when the "drop" is not to be permitted.

If the object under the mouse pointer is not the parent of the object being dragged, the default action is for the system to ignore the operation. However, by enabling the *DragDrop* event, your application can of course take whatever action is appropriate, including perhaps moving the dragged object to a new parent.

Note that a *Dragable* object does not generate a *Configure* (31) event when it is dragged and dropped.

3.91 DrawMode Property

Applies To: *Circle, Ellipse, Marker, Poly, Rect, Text*

Description

The *DrawMode* property provides direct control over the low-level drawing operation performed by graphical objects.

The *DrawMode* property specifies the current foreground mix mode. The Windows GDI uses the foreground mix mode to combine pens and interiors of filled objects with the colours already on the screen. The foreground mix mode defines how colours from the brush or pen and the colours in the existing image are to be combined.

DrawMode affects every drawing operation performed by Dyalog APL and not just the initial drawing operation when the object is created. Many of the drawing modes are additive (the result depends not just on what is being drawn, but on what is already

there) and the effects may therefore vary. For this reason, DrawMode should normally be used only with un-named graphical objects.

You could use DrawMode to move or animate graphical objects in circumstances where the standard Dyalog APL behaviour was not ideal.

DrawMode is an integer with one of the following values:

Value	Name	Resulting Pixel Colour
1	R2_BLACK	Pixel is always 0.
2	R2_NOTMERGEPEN	Pixel is the inverse of the R2_MERGEPEN colour.
3	R2_MASKNOTPEN	Pixel is a combination of the colours common to both the screen and the inverse of the pen.
4	R2_NOTCOPYPEN	Pixel is the inverse of the pen colour.
5	R2_MASKPENNOT	Pixel is a combination of the colours common to both the pen and the inverse of the screen.
6	R2_NOT	Pixel is the inverse of the screen colour.
7	R2_XORPEN	Pixel is a combination of the colours in the pen and in the screen, but not in both.
8	R2_NOTMASKPEN	Pixel is the inverse of the R2_MASKPEN colour.
9	R2_MASKPEN	Pixel is a combination of the colours common to both the pen and the screen.
10	R2_NOTXORPEN	Pixel is the inverse of the R2_XORPEN colour.
11	R2_NOP	Pixel remains unchanged.
12	R2_MERGENOTPEN	Pixel is a combination of the screen colour and the inverse of the pen colour.
13	R2_COPYPEN	Pixel is the pen colour.
14	R2_MERGEPENNOT	Pixel is a combination of the pen colour and the inverse of the screen colour.
15	R2_MERGEPEN	Pixel is a combination of the pen colour and the screen colour.
16	R2_WHITE	Pixel is always 1.

3.92 Duplex Property

Applies To: *Printer*

Description

Specifies whether pages are printed on separate sheets or back-to-back.

Duplex is a character vector which is either empty or contains 'Simplex', 'Vertical', or 'Horizontal'.

The default value for Duplex is derived from the current printer setting and 'Vertical' and 'Horizontal' are only effective if the printer supports a duplex capability.

3.93 EdgeStyle Property

Applies To

Description

This property is used to give a 3-dimensional appearance to screen objects.

This is achieved by drawing the object with a grey or white background colour and by drawing a border around it using various combinations of black, white and dark grey lines. Note that this border is drawn *outside* a control but *inside* a *Form* or *SubForm*.

Note that EdgeStyle is not honoured for objects which have a natural built-in 3-dimensional appearance.

The value of the EdgeStyle property is a character vector chosen from the following :

'None'	Object is drawn with no 3-dimensional effects and the EdgeStyle properties of its children are ignored (treated as None).
'Plinth'	Object is drawn with a light shadow along its top and left edges and a dark shadow along its bottom and right edges. This gives the illusion of a raised effect.
'Recess'	Object is drawn with a dark shadow along its top and left edges and a light shadow along its bottom and right edges. This gives the illusion of a sunken effect.
'Groove'	Object is drawn with a border that has the appearance of a groove.
'Ridge'	Object is drawn with a border that has the appearance of a ridge.
'Shadow'	Object is drawn with a dark border line along its top and left edges.
'Default'	Object itself is drawn with no 3-dimensional border, but the values of the EdgeStyle properties of its children are observed.
'Dialog'	Used in conjunction with ('Border' 2), this gives a Form the appearance of a standard 3-dimensional dialog box. This setting applies only to a <i>Form</i> or a <i>SubForm</i>

For the *Root* object, the EdgeStyle property may be 'None' or 'Default'. If EdgeStyle is 'None', screen objects are drawn without 3-dimensional effects of any kind and the value of their EdgeStyle property is ignored. If EdgeStyle is 'Default', all controls are drawn using their default EdgeStyle properties.

Note that *MsgBox*, *FileBox* and the set-up dialog box associated with the *Printer* object are all drawn with 3-dimensional effects regardless of the value of EdgeStyle on *Root*. These objects do not have their own EdgeStyle properties.

If you set EdgeStyle to 'None' on the *Root* object, all your objects will (by default) be drawn without 3-dimensional effects.

3.94 EditImage Property

Applies To: *ComboEx*

Description

Specifies whether or not the edit control portion of the *ComboEx* displays an image for selected items.

EditImage is a single number with the value 0 or 1 (the default). If EditImage is 1, the image associated with the selected item is displayed in the edit control, portion of the ComboEx object, to the left of the text. If EditImage is 0, only the item text is displayed in the edit control.

3.95 EditImageIndent Property

Applies To: *ComboEx*

Description

Specifies whether or not the indents associated with items in a ComboEx object are honoured in the edit control portion of the ComboEx.

EditImageIndent is a single number with the value 0 or 1 (the default).

If EditImageIndent is 1, the selected item which is displayed in the edit control portion of the ComboEx object is indented in the same way as when it is displayed in the dropdown portion of the object. The amount of indentation is specified by the Indents property.

If EditImageIndent is 0, the item displayed in the edit control portion of the ComboEx is not indented.

3.96 EditLabels Property

Applies To: *ListView, TreeView*

Description

The EditLabels property is Boolean and specifies whether or not the labels (specified by the Items property) in a *ListView* or *TreeView* object may be edited by the user. Its default value is 0 (editing is not allowed).

If EditLabels is 1, the user begins editing by clicking the label of the item that has the focus. This causes a pop-up edit box to appear around the item and allows the use to change it. A *BeginEditLabel* event is reported at the start of the edit operation and an *EndLabelEdit* event is reported on its completion. You may control the edit of a particular label using callback functions attached to these events.

3.97 Elevated Property

Applies To: *Button*

Description

The *Elevated* property applies only to a *Button* whose *Style* is 'CommandLink'.

Elevated is a Boolean scalar with a default value of 0. When set to 1, the icon on the CommandLink button changes from a "green arrow" to a "shield". This is intended to convey to the user that the action associated with the *Button* requires *Elevation*. This is a feature of *User Account Control* in Windows 7. See your Windows documentation for further information. Note that APL does not take any action (other than to cause the icon to change) when Elevated is set to 1. This is the responsibility of the programmer.

3.98 Encoding Property

Applies To: *TCP Socket*

Description

The Encoding property is a character vector that specifies how character data are encoded or translated.

The possible values are 'None', 'UTF-8', 'Classic', or 'Unicode', depending upon the value of the *Style* property.

Table: Unicode Edition

Style	Encoding	Description
'Raw'	'None'	Not applicable. Only integer data may be transmitted/received.
'Char'	'None'	Transmission is limited to characters with Unicode code points in the range 0-255. Attempting to transmit (or receive) a character outside this range will cause DOMAIN ERROR.
	'UTF-8'	Characters are transmitted/received using the UTF-8 encoding scheme.
'APL'	'Classic'	Characters are transmitted/received as indices of □AV, and translated according to the current value of □AVU . An attempt to transmit or receive a characters not present in □AVU will cause TRANSLATION ERROR
	'Unicode'	Characters are transmitted/received <i>as is</i> (as Unicode code points).

Table: Classic Edition

Style	Encoding	Description
'Raw'	'None'	Not applicable. Only integer data may be transmitted/received.
'Char'	'None'	Characters (which are represented internally as indices of □AV) are translated to and from ASCII using the Output Translate Table win.dot.
	'UTF-8'	Characters are converted to/from Unicode using □AVU and transmitted/received using the UTF-8 encoding scheme. An attempt to transmit or receive a characters not present in □AVU will cause TRANSLATION ERROR.
'Raw'	'Classic'	Characters are transmitted/received as indices of □AV.
	'Unicode'	Characters are converted to/from Unicode using □AVU and transmitted/received as Unicode code points. An attempt to transmit or receive a characters not present in □AVU will cause TRANSLATION ERROR.

The default value of Encoding depends upon the value of Style as indicated. Default values are highlighted thus in the above tables.

An attempt to set the value of *Encoding* to a value not valid for the current *Style*, as implied by the above tables, will cause `DOMAIN ERROR`.

If you change the value of the *Style* property, the value of *Encoding* will remain unchanged if it is valid for the new *Style*. Otherwise it will revert to the default value for the new value of *Style*.

```
's0'␣WC'TCPSocket' ('LocalPort' 2001)
s0.(Style Encoding)
Char None

s0.Style←'APL'
s0.(Style Encoding)
Apl Classic
```

3.99 End Property

Applies To: *Circle, Ellipse*

Description

This property specifies one or more end-angles for an arc, pie-slice, or chord of a circle or ellipse. It may be used in conjunction with *Start* which specifies start angles. Angles are measured counter-clockwise from the x-axis at the centre of the object.

If a single arc is being drawn, *End* is a single number that specifies the end angle of the arc in radians ($0 \leq \text{End} < 2\pi$). If multiple arcs are being drawn, *End* is either a single number as before (the end angle for several concentric arcs) or a numeric vector with one element per arc.

If *Start* is not specified, the default value of *End* is 2π . Otherwise, the default value of *End* is $(\neg 1 \downarrow + \backslash \text{Start}), 2\pi$.

3.100 EnterReadOnlyCells Property

Applies To: *Grid*

Description

This is a Boolean property that specifies whether or not the user may visit read-only cells in a *Grid* object. Its default value is 1.

In this context, a *read-only* cell is one that satisfies one or more of the following conditions:

- it has no associated *Input* object
- its associated Input object is a *Label*
- its associated Input object is an *Edit* object with *ReadOnly* set to 1
- its associated Input object is inactive (*Active* 0)

If *EnterReadOnlyCells* is set to 0 and the user clicks the mouse on a *read-only* cell, the current cell does not change although *CellDown*, *CellUp* and *CellDoubleClick* events are reported if enabled. If the user presses a cursor movement key that would otherwise cause the cursor to move into a *read-only* cell, the cursor moves instead to the nearest *editable* cell in the appropriate direction.

3.101 Event Property

Applies To

Description

This property defines how an object responds to user actions.

Unlike other properties which only have a single value, this property has a value corresponding to each of the different types of event that may be generated by a particular object. Consequently the syntax for setting the Event property differs from the general syntax that applies to other properties.

Two syntactic forms are allowed:

- A 3 or 4-item vector containing the property name 'Event', followed by the Event Type(s), a value which determines the action to be taken, and an optional array that will be supplied as a left argument to the callback function
- A composite vector whose first element contains the property name 'Event', followed by a series of 2 or 3-element vectors, each defining the action to be taken for a different Event Type (or types).

EXAMPLES

```
'Event' 'MouseUp' 'foo' 88
'Event' ('MouseUp' 'MouseDown') 'foo' 88
'Event' ('MouseUp' 'foo' 88) ('MouseDown' 'goo')
```

Like any other property, the Event property can be set using assignment. However, certain special considerations apply which are discussed later.

When you specify the Event property using `□WC` or `□WS`, the action to be taken for an event type or types is specified by a 2 or 3-element vector containing:

Element	Item	Description
[1]	Type(s)	see below
[2]	Action	numeric scalar or character vector <code>-1</code> inhibit (ignore) event <code>0</code> handle event, do not report to APL <code>1</code> handle event, then report to APL <code>f n</code> name of callback function <code>f n&</code> name of callback function to be executed <i>asynchronously</i> <code>⌘expr</code> expression to be executed
[3]	Arg	any array (optional)

Event Types

The first element, *Type(s)* may be one of the following:

- A character vector containing an event name (for example 'MouseUp')
- A numeric scalar containing an event number (for example 2). If the number is not one of the built-in event numbers generated by the object, it is assumed to be a user-defined event which can (only) be generated anagrammatically using `□NQ`
- A vector of character vectors containing a list of event names, for example ('MouseDown' 'MouseUp'). This may be used as a shortcut to associate several different types of events with the same action
- An Event name preceded by the string 'on' (for example 'onMouseUp')
- An event number preceded by the string 'on' (for example 'on99'). This syntax is intended for user-defined events although it can be used with regular events too.

The *onEvent* syntax causes all objects reported in the *event message* (see below) to be identified by a *ref*. Otherwise, objects reported in the event message are identified by name.

Action**Inhibit (-1)**

If *Action* is set to -1, the event is inhibited (if possible) by APL. If, for example, you set the action on a KeyPress event to -1, all keystrokes for the object in question will be ignored. Similarly, if you set the *action* on a Close event for a Form to -1, the user will be unable to close the Form. This is possible because APL intercepts most events before Windows itself takes any action. However, certain events (for example, focus change events) are not notified to APL until **after** the event has occurred and **after** Windows has itself responded in some way. In these circumstances it is not always practical for APL to undo what Windows has already done, and an action code of -1 is treated as if it were 0. For further details, see the individual entries for each event type in this chapter.

Default Processing (0)

If *Action* is set to 0 (the default), the event is processed by APL and Windows in the normal way (this is referred to herein as the *default processing*) but your program is not notified in any way that the event has occurred. For example, the default processing for a keystroke is to action it and either echo a character in the object or perform some other appropriate function.

Terminate □DQ (1)

If *Action* is set to 1, the event is first processed by APL (and Windows) in the normal way, then □DQ terminates, returning an *event message* as its result. The format of the event message is given under the description of each event type.

Callback (function name)

If *Action* is set to a character vector that specifies the name of a function, this function (termed a *callback*) will be executed automatically by □DQ every time the event occurs. The function may be a traditional defined function or a dfn.

A traditional defined function may be monadic, dyadic, or niladic. If dyadic, the left argument may be optional. A niladic callback may be appropriate if the function can perform its task without needing to interrogate the event message.

Unless the callback function is niladic, it will be supplied a right argument (ω for a dfn) containing the *event message* and a left argument (α for a dfn) of the value of the array *Arg* (if specified).

The function may be defined to return no result, a result, or a shy result. The result determines how the event is handled.

The *default processing* of the event is deferred until **after** the callback has been run, and may be inhibited or modified by its result. If the callback function returns no result, or returns a scalar 1, normal processing of the event is allowed to continue as soon as the callback completes. If the callback returns a scalar 0, normal processing of the event is inhibited and the effect is identical to setting *Action* to $\bar{1}$. A callback function may also return an event message as its result. If so, □DQ will action *this* event rather than the original one that fired the callback.

If a callback function does not exist at the instant it is invoked, □DQ terminates with a `VALUE ERROR`. However, the name of the missing function is reported in the Status Window.

Asynchronous Callback (function name followed by &)

If *Action* is set to a character vector that specifies the name of a callback function, followed by the character `&`, the callback function will be executed asynchronously in a new thread when the event occurs.

For example, the event specification:

```
'Event' 'onSelect' 'DoIt&'
```

tells □DQ to execute the callback function `DoIt` asynchronously as a thread when a `Select` event occurs on the object. Note that a callback function executed in this way should not return a result (because □DQ does not wait for it) and any result will be displayed in the Session window.

Execute

If action code is set to a character vector whose first element is the execute symbol ($\&$) the remaining string will be executed automatically whenever the event occurs. The default processing for the event is performed first and may not be changed or inhibited in any way.

Notice that when you specify the action to be taken on the occurrence of an event there is a great difference between 'FOO' and '⌘FOO'. The former causes APL to invoke the function FOO as a *callback function*. If the function takes an argument, APL will supply it with the event message. Secondly, the result (if any) of the function FOO will be used by APL and may cause the event to be disabled or changed in some way. In the second case, APL will perform the default processing for the event and then execute FOO without supplying an argument. If the function returns a result, it will be displayed in the Session.

Optional Left Argument (Arg)

If specified, *Arg* is an array whose value will be passed as the left argument to a callback function when that particular event (or events) is generated. Note that this is a constant defined when the value is assigned to the Event property.

If the callback function is defined to take an explicit left argument and *Arg* was not specified, the call will fail with the error message:

```
SYNTAX ERROR: The function requires a left argument
```

If the callback function is defined to take an optional left argument and *Arg* was not specified, a reference to the left argument (α for a dfn) will generate VALUE ERROR.

Event Message

When a callback function is invoked by □□□, the corresponding event message is supplied as its right argument. The event message is a vector whose first 2 elements identify the object that generated the event and the type of the event. Additional elements may be provided, depending upon the type of the event.

The same event message is returned as a (shy) result by □□□ when it is terminated by an event whose Action is set to 1.

Object(s)

The first element of the event message always identifies the object that generated the event. Other elements may identify other objects associated with the event. For example, a DragDrop event reports both the object being dropped, and the object on which it is being dropped.

Objects are identified by *names* or *refs*. If the Event property was set using the *onEvent* syntax (whereby the event name or number is prefixed by the string 'on'), for

example, 'onSelect' or 'on99', objects are identified by *refs*. This is also true if the object which generated the event has no name (that is, was created by `NEW`). Otherwise, objects are identified by their names.

Event Type

If, when the event type was specified it was identified by its name, the second element of the event message will be a character vector containing that name. If it was identified by its number, the second element of the event message will be an integer containing that number. If the event type was identified using the *onEvent* syntax, the second element of the event message will be a character vector containing the prefix 'on' followed by the event name, even if it had been specified by number. The exception is that if the event is a user-defined event, the second element of the event message will be a character vector containing the prefix 'on' followed by the character representation of the user-defined event number.

Specifying the Event property using Assignment

There are two ways to specify the Event property using assignment; you can specify the entire set of events, or you can set events one by one (see below).

To specify the entire set of events, you assign an array to the Event property. The array must contain one or more nested vectors, each containing 2 or 3 elements (*Type*, *Action* and optionally *Arg*) as described above.

Examples (F1 is a Form)

```
F1.Event ← 'onMouseDown' 'FOO'
```

Means: invoke callback function F00 on MouseDown, the first element of the right argument to F00 will contain a *namespace reference* to F1. All other events perform their default actions.

```
F1.Event ← 'MouseDown' 'FOO'
```

Means: invoke callback function F00 on MouseDown, the first element of the right argument to F00 will contain the *character vector* 'F1'. All other events perform their default actions.

```
F1.Event ← ('onMouseDown' 'FOO') ('onMouseUp' 'FOO')
```

Means: invoke callback function F00 on MouseDown and MouseUp. All other events perform their default actions.

```
F1.Event, ← = 'onMouseMove' 'F00' ( 'THIS' 1 )
```

Means: add a callback function F00 on the MouseMove event. The function will receive the array ('THIS' 1) as its left argument. All other events perform their default actions.

Specifying Individual Event types using Assignment

To define the action to be taken for individual events, one by one, you use the *onEvent* syntax and make the assignment to the event name prefixed by the string 'on'.

EXAMPLES

```
F1.onMouseDown ← 'F00'
```

Means: invoke callback function F00 on MouseDown.

```
F1.onMouseUp ← 'F00'
```

Means: add the same callback for MouseUp.

```
F1.onMouseMove ← 'F00' ( 'THIS' 1 )
```

Means: add the same callback function F00 for the MouseMove event. The function will receive the array ('THIS' 1) as its left-argument.

Notice that you must use the 'on' prefix; you cannot assign to the Event name itself. This would cause an error:

```
F1.MouseUp←'foo'
SYNTAX ERROR: Invalid modified assignment, or an attempt was made to cha
nge nameclass on assignment
F1.MouseUp←'foo'
^
```

Specifying the Event property using □WC and □WS

When you set the Event property using □WC and □WS you define the actions for the event types that you specify in the argument, leaving the actions for all other event types unchanged. When you create an object with □WC, all unspecified event types will be unhandled; that is, those events will perform the default processing. However, when

you specify the action for a new event type using `□WS`, any actions previously defined for other event types will remain as they were.

Examples using Event Names

Ignore MouseDown (1) event (APL will perform the default processing for you)

```
'F1' □WS 'Event' 'MouseDown' 0
```

Terminate `□DQ` on MouseDown

```
'F1' □WS 'Event' 'MouseDown' 1
```

Invoke callback function F00 on MouseDown, the first element of the right argument to F00 will contain a *namespace reference* to F1

```
'F1' □WS 'Event' 'onMouseDown' 'F00'
```

Invoke callback function F00 on MouseDown, the first element of the right argument to F00 will contain the *character vector* 'F1'

```
'F1' □WS 'Event' 'MouseDown' 'F00'
```

Invoke callback function F00 on MouseDown and MouseUp

```
'F1' □WS 'Event' ('onMouseDown' 'onMouseUp') 'F00'
```

Invoke callback function F00 with ('THIS' 1) as its left-argument on MouseDown

```
'F1' □WS 'Event' 'onMouseDown' 'F00' ('THIS' 1)
```

Invoke callback function F00 with ('THIS' 1) as its left-argument on MouseDown, MouseUp and MouseMove

```
EV ← 'onMouseDown' 'onMouseUp' 'onMouseMove'
'F1' □WS 'Event' EV 'F00' ('THIS' 1)
```

Execute the expression `COUNT++1` on MouseDown

```
'F1' □WS 'Event' 'MouseDown' '⍎COUNT++1'
```

Execute the expression `COUNT++1` on MouseDown, MouseUp and MouseMove

```
EV ← 'MouseDown' 'MouseUp' 'MouseMove'
'F1' □WS 'Event' EV '⍎COUNT++1'
```

Examples using Event Numbers

Ignore MouseDown (1) event (APL will perform the default processing for you)

```
'F1' □WS 'Event' (1 0)
'F1' □WS 'Event' 1 0      A Ditto
```

Terminate □DQ on MouseDown

```
'F1' □WS 'Event' (1 1)
'F1' □WS 'Event' 1 1      A Ditto
```

Call function F00 on MouseDown

```
'F1' □WS 'Event' (1 'FOO')
'F1' □WS 'Event' 1 'FOO'    A Ditto
```

Call function F00 on MouseDown and MouseUp

```
'F1' □WS 'Event' ((1 2) 'FOO')
'F1' □WS 'Event' (1 2) 'FOO'      A Ditto
'F1' □WS 'Event' 1 2 'FOO'        A Ditto
'F1' □WS 'Event' (1 'FOO')(2 'FOO') A Ditto
```

Call function F00 with ('THIS' 1) as its left-argument on MouseDown

```
'F1' □WS 'Event' (1 'FOO' ('THIS' 1))
'F1' □WS 'Event' 1 'FOO' ('THIS' 1) A Ditto
```

Call function F00 with ('THIS' 1) as its left-argument on MouseDown and MouseUp

```
'F1' □WS 'Event' ((1 2) 'FOO' ('THIS' 1))
'F1' □WS 'Event' (1 2) 'FOO' ('THIS' 1) A Ditto
'F1' □WS 'Event' 1 2 'FOO' ('THIS' 1)   A Ditto
'F1' □WS 'Event' 1 2 'FOO' ('THIS' 1)   A Ditto
```

Execute the expression COUNT++1 on MouseDown

```
'F1' □WS 'Event' 1 '⊔COUNT++1'
```

Execute the expression COUNT++1 on MouseDown, MouseUp and MouseMove

```
'F1' □WS 'Event' (1 2 3) '⊔COUNT++1'
'F1' □WS 'Event' 1 2 3 '⊔COUNT++1'    A Ditto
```


User defined Events

In addition to the standard events supported directly by Dyalog APL, you may specify your own events. For these, you **must** use event numbers; user-defined event names are not allowed.

You may use any numbers not already defined, but it is strongly recommended that you choose numbers greater than 1000 to avoid potential conflict with future releases of Dyalog APL.

You can only **generate** user-defined events under program control with `⎕NQ`.

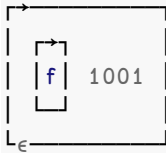
EXAMPLES

```

    ▽ foo m
[1]   ⎕SE.UCMD'display m'
    ▽

    'f' ⎕WC'Form' ('Event' 1001 'foo')
    f.Event
1001  #.foo
    ⎕NQ 'f' 1001

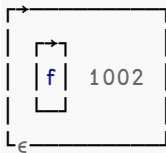
```



```

    'f' ⎕WS 'Event' 1002 'foo'
    f.Event
1001  #.foo  1002  #.foo
    ⎕NQ 'f' 1002

```



Notice that if you use the `onEvent` syntax, the event property reports the event type as you specified, but the callback function receives just the number as before.

```

f.on1003←'foo'
f.Event
1001 #.foo 1002 #.foo on1003 #.foo
[NQ 'f' 1003
┌───────────┐
│ #.f 1003 │
└───────────┘

```

Notes

Resetting (clearing) the Event Property

If no events are set, the result obtained by `WG` and the result obtained by referencing `Event` directly are different:

```

'F'[WG'Form'
DISPLAY 'F'[WG'Event'

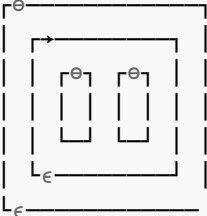
```



```

DISPLAY F.Event

```



To reset the Event property, the same (different) values must be used accordingly:

```
f.Event←0p<' ' ' '
```

or

```
'f'[WS'Event' 0 0
```

onEvent Syntax with Event Numbers

If you use the `onEvent` syntax with *built-in* event numbers, the effect is the same as if you had used the event name. This does not apply to user-defined events.

EXAMPLE

```

    'f' □WC 'Form'
    f.on2 ← 'foo'
    f.Event
    onMouseUp #.foo
    ▼foo▼
    ▼ foo m
[1]   □SE.UCMD 'display m'
    ▼

```

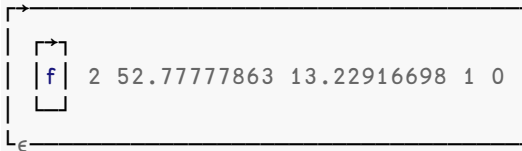


This differs from the behaviour when you use event number normally:

```

    'f' □WC 'Form'
    'f' □WS 'Event' 2 'foo'
    f.Event
    2 #.foo

```



Callback Names

When you query the Event property using □WG, names of callbacks associated with events are reported exactly as they were set. When you reference the Event property, the names are reported as absolute pathnames.

```

)NS x
#.x
)CS x
#.x
'f'␣WC'form'
f.onMouseUp←'foo'
f.Event

|onMouseUp|#.x.foo|

'f'␣WC'event'

|onMouseUp|foo|

)CS
#
#.x.f.Event

|onMouseUp|#.x.foo|

'#.x.f'␣WC'Event'

|onMouseUp|x.foo|

```

Spelling Event Names

When using regular event names, case is unimportant. For example, the system will accept 'MouseUp', 'MOUSEUP' or even 'mOuSeUp'. When using the *onEvent* syntax, case is critical. The 'on' must be in lower-case and the case of the event name must be spelled exactly as documented. In all cases, the event name will be reported using the documented spelling.

Special Case for All events

The event number 0 and the event name 'All', are convenient shorthands to associate a particular action (such as a callback) with **all** the built-in events supported by an object.

EXAMPLE

```
'f' WC 'Form' ( 'Event' 'All' 1 )
f.Event
All 1
WC 'f'
f Create 1
```

3.102 EventList Property

Applies To

Description

This is a read-only property that reports the names of all the events supported by an object.

3.103 ExportedFns Property

Applies To: *OLEServer*

Description

This property specifies the functions to be exposed as methods by an *OLEServer* object.

ExportedFns may be set to 0 (none), 1 (all), or a vector of character vectors containing the names of the functions to be exported.

There are certain important restrictions concerning the type of function that you can export as a method.

Firstly, only top-level defined functions within the *OLEServer* may be exported; you cannot export functions in other namespaces including sub-namespaces.

Furthermore, you may not export defined operators, dfns, external functions, or functions created by function assignment.

Finally, OLE does not support the concept of a dyadic function, so your exported functions must be niladic, monadic, or take an optional left argument; they may not be explicitly dyadic.

If you wish to export a new function from your *OLEServer*, and *ExportedFns* is not 1, you must explicitly reset the value of the *ExportedFns* property before you re-save the workspace.

3.104 ExportedVars Property

Applies To: *OLEServer*

Description

This property specifies the variables to be exposed as properties by an *OLEServer* object.

ExportedVars may be set to 0 (none), 1 (all), or a vector of character vectors containing the names of the variables to be exported.

Note that you may not export external variables or shared variables, or variables in other namespaces.

If you wish to export a new variable from your *OLEServer*, and *ExportedVars* is not 1, you must explicitly reset the value of the *ExportedVars* property before you re-save the workspace.

3.105 FCol Property

Applies To

Description

This property defines the foreground colour(s) of an object.

For non-graphical objects, *FCol* specifies a single colour. For graphical objects with more than one constituent part, it may specify a set of background colours, one for each part. A single colour is represented by a single number which refers to a standard colour, or by a 3-element vector which defines a colour explicitly in terms of its red, green and blue intensities.

If *FCol* is 0 (which is the default) the foreground colour is defined by your current colour scheme for the object in question. For example, if you select red as your MS-Windows

"Button Text" colour, you will by default get red writing on all your *Button* objects, simply by not specifying FCol or by setting it to 0.

A negative value of FCol refers to a standard MS-Windows colour as described below. Positive values are reserved for a possible future extension.

FCol	Colour Element	FCol	Colour Element
0	Default	-11	Active Border
-1	Scroll Bars	-12	Inactive Border
-2	Desktop	-13	Application Workspace
-3	Active Title Bar	-14	Highlight
-4	Inactive Title Bar	-15	Highlighted Text
-5	Menu Bar	-16	Button Face
-6	Window Background	-17	Button Shadow
-7	Window Frame	-18	Disabled Text
-8	Menu Text	-19	Button Text
-9	Window Text	-20	Inactive Title Bar Text
-10	Active Title Bar Text	-21	Button Highlight

If instead, FCol contains a 3-element vector, it specifies the intensity of the red, green and blue components of the colour as values in the range 0-255. For example, (255 0 0) is red and (255 255 0) is yellow.

Note that if the colour specified by FCol would normally be rendered as a dithered colour, it is instead converted to the nearest pure colour available on the device. The actual colour realised also depends upon the capabilities of the display adapter and driver, and the current Windows colour map.

For a *Button*, *Combo*, *Edit*, *Label*, *List*, *Menu* and *MenuItem*, FCol refers to the colour of the **text** in the object. *Borders* around these objects (where applicable) are drawn using the standard Windows colour. For a *Static* object however, FCol specifies the colour of its border.

For the *Ellipse*, *Poly* and *Rect* objects, FCol specifies the colour of the line drawn around the perimeter of the object. If a dashed or dotted line is used (*LStyle* 1-4) the "gaps" in the line are drawn using the colour specified by *BCol*, or are left undrawn if *BCol* is not specified. For the *Marker* object, FCol specifies the colour in which the markers are

drawn. For the graphics objects *Ellipse*, *Poly*, *Rect* and *Text*, *FCol* may be a vector of 3-element vectors specifying a set of colours for the constituent parts of the object. For example, a *Poly* object consisting of four polygons, may have a *FCol* property of four 3-element vectors. In addition, for graphics objects, *FCol* is used in place of *FillCol* if the latter is not specified.

3.106 FStyle Property

Applies To: *Circle*, *Ellipse*, *Poly*, *Rect*

Description

This property determines how a graphics object is filled. It takes one of the following values, or, if the object has more than one component, a vector of such values.

FStyle	Effect
-1	hollow (no fill). This is the default.
0	solid fill
1	hatch fill with horizontal lines
2	hatch fill with vertical lines
3	hatch fill with diagonal lines at 135 degrees
4	hatch fill with diagonal lines at 45 degrees
5	hatch fill with horizontal and vertical lines
6	hatch fill with criss-crossing diagonal lines
str	the name of, or a ref to, a <i>Bitmap</i> object which is used to fill the object.

For example, to fill an object with criss-crossing diagonal lines you would specify (`'FStyle' 6`). If the object contained two components, you could fill the first one with criss-crossing diagonal lines, and the second one with a *Bitmap* called 'YES', with the specification (`'FStyle' 6 'YES'`)

If the size of the *Bitmap* is 8x8 APL uses a Windows "brush" to fill the object. If not, it uses "tiling". Filling with a brush is faster.

3.107 FieldType Property

Applies To: *ButtonEdit*, *DateTimePicker*, *Edit*, *Label*, *Spinner*

Description

The `FieldType` property controls data conversion, formatting and validation.

For *Edit*, *Label* and *Spinner* objects, `FieldType` controls how the *Value* property of these objects is interpreted.

`FieldType` is a character vector. If it is empty (the default), the *Text* or *Label* object is a standard text object with no special formatting or, in the case of an *Edit*, no input validation. For a *DateTimePicker*, an empty `FieldType` implies the default which is `'Date'`.

For a *DateTimePicker*, `FieldType` may be one of the following:

<code>'Date'</code>	Uses Windows "short date" format
<code>'DateCentury'</code>	Uses Windows "short date" format but with a 4-digit year regardless of user preference
<code>'LongDate'</code>	Uses Windows "long date" format
<code>'Time'</code>	Uses Windows time format
<code>'Custom'</code>	Uses a special format defined by the <code>CustomFormat</code> property

The value of the date or time is represented by the *DateTime* property. Note that all validation is performed by the object itself, and it is impossible to enter an invalid value.

For an *Edit*, *Label* and *Spinner*, if `FieldType` is defined, the contents of the object are defined by its *Value* property, which is a number, rather than by its *Text* property, and special formatting and validation rules are applied. `FieldType` may be one of the following :

<code>'Numeric'</code>	Simple numeric formatting and validation
<code>'LongNumeric'</code>	Uses Windows number format
<code>'Date'</code>	Uses Windows "short date" format
<code>'LongDate'</code>	Uses Windows "long date" format
<code>'Currency'</code>	Uses Windows currency format
<code>'Time'</code>	Uses Windows time format
<code>'Char'</code>	No validation, forces <i>Value</i> to be a character vector

FieldType 'Char' only affects an *Edit* object. When the user enters data into a standard single-line *Edit* object, the *Value* property is set to a number if the contents are numeric, or to a character vector if the contents do not represent a valid number. If FieldType is 'Char', the *Value* property is always set to a character vector, regardless of the type of the field contents.

If FieldType is 'Numeric', the object displays the number defined by its *Value* property rounded to the number of decimal places specified by its *Decimals* property. The decimal separator character used will be as specified by the Number format in the user's International Control Panel settings. If the object is an *Edit* object, the user is prevented from entering anything but a valid number. The number of decimal digits is also restricted to *Decimals*. When the user "leaves" the object, the number is re-formatted.

If FieldType is 'LongNumeric', the object displays the number specified by its *Value* property according to the Number format in the user's International Control Panel settings. This format specifies the 1000 separator, decimal separator, decimal digits and whether or not a leading zero is inserted. If the object is an *Edit* object, the user is prevented from entering anything but a valid number. However, the character specified for the 1000 separator is ignored and may be entered anywhere in the number. When the user "leaves" the object, the number is re-formatted correctly.

If the FieldType is 'Currency', the object displays the number specified by its *Value* property according to the Currency format in the user's International Control Panel settings. This specifies the currency symbol and placement, the way in which a negative value is displayed, and the number of decimal places. If the object is an *Edit* object, the user is restricted to entering a "reasonable" value. When the user leaves the object, the number is reformatted correctly.

If the FieldType is 'Date', the *Value* property represents the number of days since January 1st 1900 and is displayed using the "short date" format specified by the user's International Control Panel settings. If the object is an *Edit* object, the user is restricted to entering a "reasonable" date. The object will accept any numeric triplet separated by slash(/), colon(:) or space characters but checks that the day number and month number lie in the range 1-31 and 1-12 respectively and will not allow the user to enter a digit that would invalidate this. (Note that the position within the triplet of the day, month and year are as specified by the Windows short date format). However, the user is not prevented from entering an invalid date such as 31st September.

If the FieldType is 'LongDate', the *Value* property represents the number of days since January 1st 1900 and is displayed using the "long date" format specified by the user's International Control Panel settings. If the object is an *Edit* object, its appearance and behaviour automatically switches to FieldType 'Date' when it has the input focus and

back again when it loses the focus. This allows the user to edit or input a date in a more convenient form.

If the *FieldType* is 'Time', the *Value* property represents the number of seconds since midnight and is displayed using the time format specified by the user's International Control Panel settings.

When the user attempts to move the input focus away from the object, the contents are validated. If they cannot be converted to a valid number, date, or time, the object generates a *BadValue* event, or, if the object is associated with a *Grid*, the *Grid* (and not the *Edit* object) generates a *CellError* event. See the descriptions of these events for further details.

Note that for *Edit*, *Label* and *Spinner* objects, *FieldType* may only be specified when you create an object using `□WC`.

3.108 File Property

Applies To

Description

Specifies the name of a file associated with an object.

For an *Animation*, *Bitmap*, *Cursor* or *Icon* object, this property is either a simple character vector or a 2-element nested vector.

If it is simple, *File* specifies the name of the associated bitmap (.BMP), icon (.ICO) or cursor (.CUR) file.

If it is nested, the first element specifies the name of a DLL or (*Icon* only) EXE and the second element identifies the particular bitmap, icon or cursor in that file. The identifier may be its *name* (a character string), its *resource id* (a non-zero positive integer) or (*Icon* only), its *index* (0 or negative integer) within the file. As a special case, if the name of the file is an empty vector, the object is loaded from DYALOG.EXE or the Dyalog resource DLL. The name of the latter varies according to the version installed but begins DYARES. In this case, the identifier must be a name or resource id; indexes are not supported.

For a *Metafile* object, *File* must be simple and specifies the name of a metafile (.WMF) file. For a *RichEdit* object, *File* must be simple and specifies the name of a Rich text Format (.RTF) file.

When applied to a *FileBox* object, *File* contains the name (or names) of the selected file (or files) depending upon the value of its *Style* ('*Single*' or '*Multi*').

3.109 FileMode Property

Applies To: *FileBox*

Description

The *FileMode* property applies only to a *FileBox* object. It indicates the mode in which the selected file is going to be opened. It is a character vector containing '*Read*' (the default) or '*Write*'. If *FileMode* is '*Write*', files listed in the File Selection Box are greyed, although they may still be selected.

3.110 FillCol Property

Applies To: *Circle*, *Ellipse*, *Poly*, *Rect*

Description

This property defines the fill colour in a graphics object.

If *FStyle* is 0 (solid fill) *FillCol* defines the colour with which the object is filled. If *FStyle* is in the range 1-6 (pattern fill) it defines the colour of the lines that make up the pattern. The areas between the lines are filled using the colour specified by *BCol*, or are left undrawn (transparent) if *BCol* is not specified. If *FStyle* contains the name of a *Bitmap* object, the value of *FillCol* is ignored.

A single colour is represented by a single number which refers to a standard colour, or by a 3-element vector which defines a colour explicitly in terms of its red, green and blue intensities. A negative value of *FillCol* refers to a standard MS-Windows colour as described below. Positive values are reserved for a possible future extension.

FillCol	Colour Element	FillCol	Colour Element
0	Default	-11	Active Border
-1	Scroll Bars	-12	Inactive Border
-2	Desktop	-13	Application Workspace
-3	Active Title Bar	-14	Highlight
-4	Inactive Title Bar	-15	Highlighted Text
-5	Menu Bar	-16	Button Face
-6	Window Background	-17	Button Shadow
-7	Window Frame	-18	Disabled Text
-8	Menu Text	-19	Button Text
-9	Window Text	-20	Inactive Title Bar Text
-10	Active Title Bar Text	-21	Button Highlight

If instead, FillCol contains a 3-element vector, it specifies the intensity of the red, green and blue components of the colour as values in the range 0-255. For example, (255 0 0) is red and (255 255 0) is yellow. Note that the colour realised depends upon the capabilities of the display adapter and driver, and the current Windows colour map.

FillCol may also be a vector of 3-element vectors specifying a set of colours for the constituent parts of the object. For example, a *Poly* object consisting of four polygons, may have a FillCol property of four 3-element vectors.

3.111 Filters Property

Applies To: *FileBox*

Description

The Filters property is a nested scalar or vector containing a list of filters. Each filter is a 2-element vector of character vectors which contain a file type mask and a file type description respectively. The file type descriptions appear in a drop-down combo box labelled "List Files of Type". When the user selects one of these, the currently selected directory is searched for files which match the corresponding mask. The default value of Filters is an empty vector. This gives a file type mask of "." and a file type description of "All Files (.)". Hence an empty vector is equivalent to `(c'*. *' 'All Files (*.*)')`.

3.112 FireOnce Property

Applies To: *Timer*

Description

This property specifies one-off behaviour for a *Timer* object. It has the value 0, 1 or 2.

Setting FireOnce to 1, will cause the *Timer* to generate a single event and no more unless it is reset. After generating the single event, FireOnce is automatically set to 2 and this change occurs prior to the invocation of a callback function.

Setting FireOnce to 2 will cause the *Timer* to behave as if it has already raised its single event.

FireOnce honours the value of the *Active* property but does not change it. So setting FireOnce to 1 when *Active* is 0 will not immediately cause an event. If *Active* is subsequently set to 1, the single *Timer* event will then occur.

3.113 FirstDay Property

Applies To: *Calendar*

Description

The FirstDay property specifies the day that is considered to be the first day of the week and which appears first in the *Calendar*.

FirstDay is an integer whose value is in the range 0-6. The default value for FirstDay depends upon your International Settings, but in most countries is 0 meaning Monday.

3.114 Fixed Property

Applies To: *Font*

Description

This property specifies whether or not a font represented by a *Font* object is fixed-width or proportional. It is either 0 (fixed-width) or 1 (proportional). There is no default; the value of this property reflects the characteristic of the selected font.

3.115 FixedOrder Property

Applies To: *CoolBar*

Description

The FixedOrder property specifies whether or not the *CoolBar* displays *CoolBands* in the same order.

FixedOrder is a single number with the value 0 (user may re-order bands) or 1 (user may not re-order bands); the default is 0.

If FixedOrder is 1, the user may move bands to different rows, but the band order is static.

3.116 FlatSeparators Property

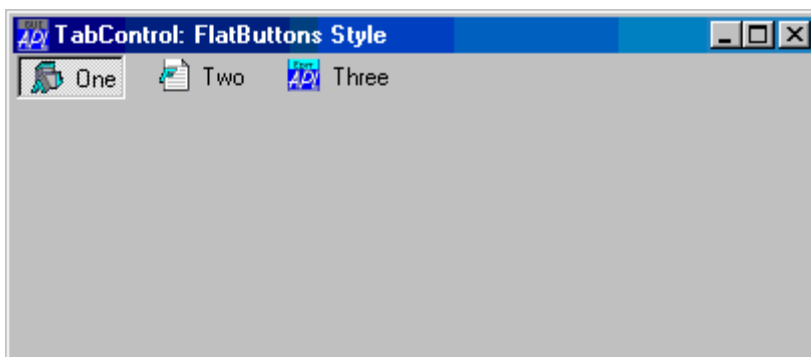
Applies To: *TabControl*

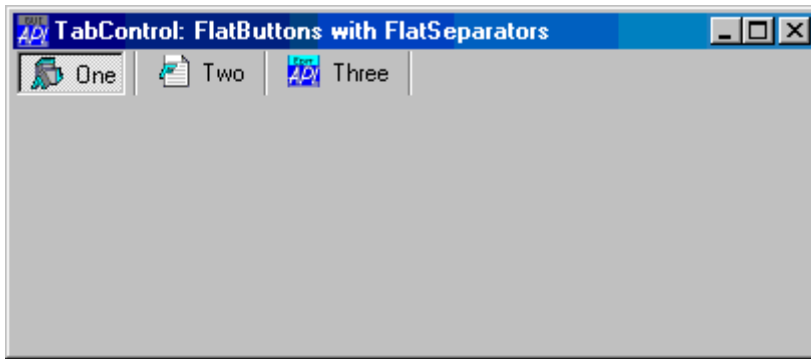
Description

The FlatSeparators property specifies whether or not separators are drawn between buttons in a *TabControl* object. FlatSeparators only affects a *TabControl* if *Style* is 'FlatButtons' and is otherwise ignored.

FlatSeparators is a single number with the value 0 (no separators) or 1 (separators); the default is 0.

The pictures below illustrate the effect of FlatSeparators on the appearance of a *TabControl* object.





3.117 FontList Property

Applies To: *Printer, Root*

Description

The FontList property is a read-only property (you cannot set its value) that provides a list of available fonts.

Its value is a vector (1 per font) of 8-element character vectors, each of which is as follows :

[1]	Face name (character vector)
[2]	Character height in "points" (integer)
[3]	Fixed width or not (Boolean)
[4]	Italic or not (Boolean)
[5]	Underline or not (Boolean)
[6]	Weight (integer)
[7]	Angle of rotation (integer)
[8]	Character set (see <i>CharSet</i>)

EXAMPLE

↑'. 'WG'FontList'							
System	16	0	0	0	700	0	0
Terminal	12	1	0	0	400	0	255
Fixedsys	15	1	0	0	400	0	0
Roman	37	0	0	0	400	0	255
Script	36	0	0	0	400	0	255
Modern	37	0	0	0	400	0	255
Small Fonts	3	0	0	0	400	0	0
Courier	13	1	0	0	400	0	0
Serif	13	0	0	0	400	0	0
Dyalog Alt	16	1	0	0	400	0	0
Dyalog Std	16	1	0	0	400	0	0

Note that the list of fonts obtained from FontList for a *Printer* object will include TrueType fonts and printer fonts but will exclude screen fonts. FontList for *Root* will include TrueType fonts and screen fonts, but exclude printer-only fonts. The two lists will therefore (typically) be different.

3.118 FontObj Property

Applies To

Description

The FontObj property associates a font with an object. It specifies either the name of or a ref to a *Font* object

If unspecified, the default value for FontObj is an empty character vector. For most objects, this setting implies that the font used in the object is **inherited** from its parent object. However, *CoolBar*, *Menu*, *MenuBar*, *StatusBar*, *TipField*, *ToolBar*, and *ToolControl* objects do not inherit their font.

Note that the default value of FontObj for *Root* is also an empty character vector and that this implies the Windows default GUI font, which is a Windows user preference setting.

Note however that it is not currently possible to specify the font for *Menu* and *MenuItem* objects which are the direct descendants of a *MenuBar*. Nor is it possible to specify the font used for the *Caption* in a *Form*.

3.119 FormatString Property

Applies To: *ButtonEdit, Edit, Grid, Label, Spinner*

Description

The `FormatString` property specifies one or more `%FMT` format specifications to be used to format data in an `Grid` or single-line `Edit`. When applied to a `Grid`, it is either a simple character vector that specifies the format specification for the entire `Grid`, or a vector of character vectors. If it is a vector, its elements are mapped to individual cells via the `CellTypes` property. When applied to an `Edit` object, `FormatString` must be a simple character vector.

APL derives the text to be displayed in a cell by calling `%FMT` with a left argument of the corresponding element of `FormatString` and a right argument of the cell value. If the format specification is invalid, the text displayed is blank.

When a formatted `Edit` object receives the focus, it redisplay the contents in its raw (unformatted) form. When the `Edit` loses the focus, its contents are reformatted. When the user moves to a formatted `Grid` cell, the text remains formatted until the user presses a non-movement key or enters *in-cell* mode. The data is then redisplayed in its raw form for editing. Data in the cell is reformatted when the user moves away.

In a `Grid`, formatted data may be aligned vertically using the `AlignChar` property as illustrated in the following example.

```
'F'⎕WC'Form'
'F.G'⎕WC'Grid'('50+?10 10p100')(0 0)(100 100)
'F.G'⎕WS'FormatString' 'M<(>N<)>F12.3'
'F.G'⎕WS'AlignChar' '.'
```

	A	B	C
1	[36.000]	26.000	[4.000]
2	2.000	34.000	[46.000]
3	19.000	9.000	44.000
4	27.000	[23.000]	[45.000]
5	49.000	23.000	26.000
6	[2.000]	[26.000]	[22.000]
7	1.000	2.000	[18.000]
8	[11.000]	[22.000]	42.000
9	[37.000]	[48.000]	19.000
10	[19.000]	[14.000]	2.000

3.120 Formats Property

Applies To: *Clipboard*

Description

This is a "read-only" property that identifies the formats in which data is currently available in the clipboard. It is a vector of character vectors containing the names of the corresponding *Clipboard* properties for which data may be obtained using *Dyalog APL Language: Wg*. In the following example data was copied to the Windows clipboard from Microsoft Excel. This product stores data in CF_Text and the older device-dependent CF_Bitmap formats. The latter excludes colour map information, so CMap is not available.

```
'CL' ⍵WC 'Clipboard'
'CL' ⍵WC 'Formats'
Bits CMap Text
```

3.121 FullRowSelect Property

Applies To: *ListView, TreeView*

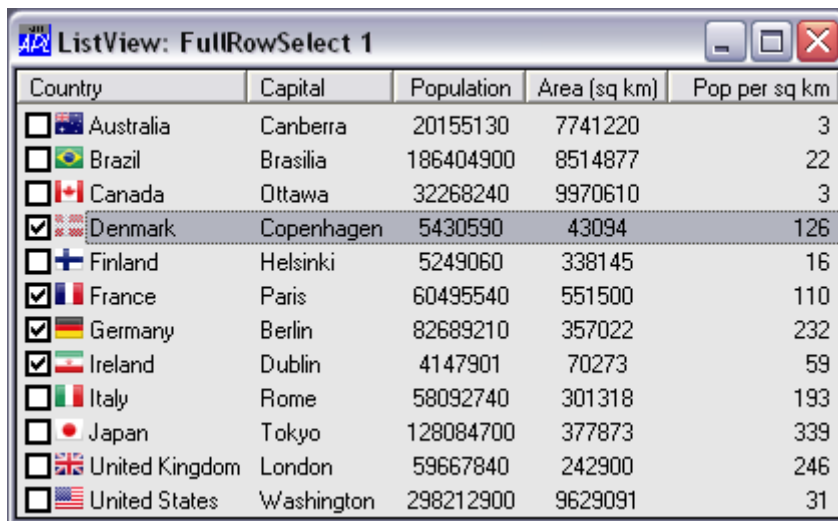
Description

The FullRowSelect property specifies whether or not the entire row is highlighted when an item in a *ListView* or a *TreeView* is selected.

FullRowSelect is a single number with the value 0 (only the item name is highlighted) or 1 (the whole row is highlighted); the default is 0.

For a *ListView*, FullRowSelect only applies if its View property is set to 'Report'.

The picture below illustrates the effect on the appearance of a *ListView* object, of setting FullRowSelect to 1.



Country	Capital	Population	Area (sq km)	Pop per sq km
☐ Australia	Canberra	20155130	7741220	3
☐ Brazil	Brasilia	186404900	8514877	22
☐ Canada	Ottawa	32268240	9970610	3
☒ Denmark	Copenhagen	5430590	43094	126
☐ Finland	Helsinki	5249060	338145	16
☒ France	Paris	60495540	551500	110
☒ Germany	Berlin	82689210	357022	232
☒ Ireland	Dublin	4147901	70273	59
☐ Italy	Rome	58092740	301318	193
☐ Japan	Tokyo	128084700	377873	339
☐ United Kingdom	London	59667840	242900	246
☐ United States	Washington	298212900	9629091	31

3.122 GridBCol Property

Applies To: *Grid*

Description

This property specifies the colour used to fill the area between the end of the last column of data and the right edge of the *Grid* and between the bottom row of data and the bottom edge of the *Grid*.

GridBCol may be a 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or it may be a scalar that defines a standard Windows colour element (see *BCol* for details). Its default value is 0 which obtains the colour defined for Window Background.

3.123 GridFCol Property

Applies To: *Grid*

Description

The GridFCol property specifies the colour of the grid lines in a *Grid* object.

GridFCol may be a 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or it may be a scalar that defines a standard Windows colour element (see *BCol* for details). Its default value is 0 which obtains the colour defined for Window text.

The grid lines may be removed by setting GridFCol to the same colour as the background colour of the cells, which is defined by *BCol*.

Finer control of the colour of the grid lines can be achieved by using *GridLineFCol* instead.

Unlike *GridLineFCol*, the value of GridFCol is ignored when *Native Look and Feel* is enabled; the colour is taken from the current theme.

3.124 GridLineFCol Property

Applies To: *Grid*

Description

The GridLineFCol property specifies the colours of the grid lines in a *Grid* object. GridLineFCol should be used if different coloured grid lines are required. If all the grid lines are the same colour, use *GridFCol*.

GridLineFCol may be a scalar or a vector. Each item may be a 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or a scalar that defines a standard Windows colour element (see *BCol* for details). Note that a single RGB triplet must be enclosed.

The default value of GridLineFCol is an empty numeric vector ($\emptyset$). If so, all the grid lines are drawn using the single colour specified by *GridFCol*.

Unlike *GridFCol*, setting GridLineFCol overrides the colour which would be used if *Native Look and Feel* was enabled.

Elements of `GridLineFCol` are allocated to individual grid lines via the *RowLineTypes* and *ColLineTypes* properties.

See also: *GridLineWidth*.

3.125 GridLineWidth Property

Applies To: *Grid*

Description

The `GridLineWidth` property specifies the widths in pixels of the grid lines in a *Grid* object.

`GridLineWidth` may be an integer scalar or a vector. Its default value is an empty numeric vector ($\emptyset$). If so, grid lines are drawn 1-pixel wide.

Grid lines are always displayed so that 1 pixel is drawn *within* the cell. If the width is greater than 1 pixel, the additional pixels are drawn *between* the cells.

If an element of `GridLineWidth` is 0, the corresponding grid lines are not drawn.

Elements of `GridLineWidth` are allocated to individual grid lines via the *RowLineTypes* and *ColLineTypes* properties.

See also: *GridLineFCol*.

3.126 GridLines Property

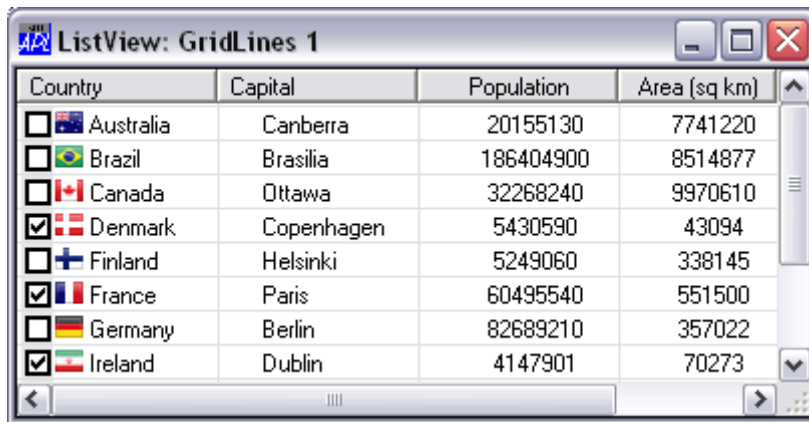
Applies To: *ListView*

Description

The `GridLines` property specifies whether or not lines are displayed between items in a *ListView* object. `GridLines` applies only if the value of the *View* property is 'Report'.

`GridLines` is a single number with the value 0 (no lines are displayed) or 1 (lines are displayed); the default is 0.

The picture below illustrate the effect on the appearance of a *ListView* object, of setting `GridLines` to 1.



3.127 GripperMode Property

Applies To: *CoolBand*

Description

The *GripperMode* property specifies whether or not the *CoolBand* has a gripper bar which is used to reposition and resize the *CoolBand* within its parent *CoolBar*.

GripperMode is a character vector with the value 'Always' (the default), 'Never' or 'Auto'.

If *GripperMode* is 'Always', the *CoolBand* displays a gripper bar even if it is the only *CoolBand* in the *CoolBar*.

If *GripperMode* is 'Never', the *CoolBand* does not have a gripper bar and may not be directly repositioned or resized by the user.

If *GripperMode* is 'Auto', the *CoolBand* displays a gripper bar only if there are other *CoolBands* in the same *CoolBar*.

3.128 HAlign Property

Applies To: *Text*

Description

This property determines the horizontal alignment of text in a *Text* object. It is either a single integer value, or, if the *Text* object contains several components, a corresponding vector of such values.

These may be:

0	left aligned	the left edge of the bounding box of the text is aligned on the x-coordinate specified by the <i>Points</i> property.
1	centre aligned	the centre of the bounding box of the text is aligned on the x-coordinate specified by the <i>Points</i> property.
2	right aligned	the right edge of the bounding box of the text is aligned on the x-coordinate specified by the <i>Points</i> property.

3.129 HScroll Property

Applies To

Description

For most objects to which it applies, this property determines whether or not a horizontal scrollbar is provided.

HScroll may only be set when the object is created with `WC` and may not subsequently be changed with `WS` or assignment.

When applied to a *Combo*, or to an *Edit* object with *Style* 'Single' (that is, a single-line edit field), the value 0 inhibits scrolling, and prevents the user from entering more data when the field is full. If instead it has the value `-2`, the field is scrollable, and the length of data that may be entered is not limited by the length of the field.

When applied to an *Edit* object with *Style* 'Multi' (that is, a multi-line text box), the value 0 inhibits scrolling, and causes individual lines to be "word-wrapped". The values `-2` and `-1` enable sideways scrolling, and permit individual lines to exceed the width of the object. The value `-1` means that a horizontal scrollbar is provided.

For a *Scroll* object, the scrollbar is horizontal if HScroll is `-1` and vertical if HScroll is 0. For a *Form*, a horizontal scrollbar is provided if HScroll is set to `-1`. The default value is 0 (no scrollbar).

For a *StatusBar*, *TabBar* or *ToolBar* with *Align* set to Top or Bottom, HScroll determines whether or not a horizontal scrollbar is provided and how the object positions its children. If HScroll is 0 (the default) the object organises its children in multiple rows

and does not provide a scrollbar. If `HScroll` is `-1` or `-2`, the object organises its children in a single row and provides a mini scrollbar to allow those positioned beyond the right edge of the object to be scrolled into view. If `HScroll` is `-1`, the scrollbar is always shown. If `HScroll` is `-2`, it is only shown when needed.

For a *Grid*, `HScroll` may be `0` (no horizontal scrollbar), `-1` (scrollbar is displayed when required), `-2` (same as `-1`) or `-3` (scrollbar is always displayed).

3.130 HTML Property

Applies To: *HTMLRenderer*

Description

The `HTML` property is a character vector that specifies the content to be rendered by the *HTMLRenderer* object. Dyalog does not perform any pre-processing of the text nor does it verify that it is properly formed HTML using single-byte character data, including any necessary escaping and encoding.

See also: *URL*.

Note

Typically, you will need to UTF-8 encode any text outside the Unicode range 0-127.

3.131 Handle Property

Applies To

Description

This is a read-only property that reports the *handle* associated with an object. For a visual object, such as a *Form* or a *Button*, this is the window handle. For a *Printer*, it is the *printer device context*.

This handle allows you to access the corresponding object directly with Windows API functions via `□NA`. This facility must be used with care and the responsibility for its

behaviour is entirely yours. Do NOT use it to delete an object. This will cause APL to crash.

An example of the use of the Handle property is to set tab stops in a *List* object. This is illustrated by the following function:

```

▼ obj TABSTOPS stops;I;LB_SETTABSTOPS;SetTabStops;sink;args
[1]  A Sets the tabstops in the List Box OBJ to be at
[2]  A stops Horizontal Dialog units.
[3]  A Sends LB_SETTABSTOPS (402) to the List Box
[4]  A See Windows SDK for Details.
[5]
[6]  I←obj □WG'Items'
[7]
[8]  LB_SETTABSTOPS←402
[9]  'SetTabStops'□NA'U4 USER32|SendMessageA U4 U4 U4 <U4[]'
[10]
[11]  args←(obj □WG'Handle') LB_SETTABSTOPS (ρ,stops)(,stops)
[12]  sink←SetTabStops args
[13]
[14]  obj □WS'Items'I
▼

```

3.132 HasApply Property

Applies To: *PropertySheet*

Description

The HasApply property is a Boolean value that specifies whether or not a *PropertySheet* has an Apply button. Its default value is 1. Note that an Apply button is only actually used if Style is 'Standard'.

3.133 HasButtons Property

Applies To: *TreeView*

Description

The HasButtons property is a Boolean value and specifies whether or not buttons are shown in a *TreeView* object. If HasButtons is 1 (the default) a square button is displayed to the left of each parent item label. If the item is expanded (that is, its children are visible) the button contains a minus sign. If the item is not expanded, (that is, its

children are hidden) the button contains a plus sign. The user can cause a parent item to expand or collapse by clicking this button.

3.134 HasCheckBox Property

Applies To: *DateTimePicker*

Description

Specifies whether or not a checkbox is displayed alongside the value in a *DateTimePicker*.

HasCheckBox is a single number with the value 0 (the default) or 1. If *HasCheckBox* is 1, the user may set or clear the checkbox to indicate whether or not the date/time displayed in the object is to apply.

If the checkbox is not set, the *DateTimePicker* is considered to be *empty* (the contents will be grayed out) and the value returned by the *DateTime* property is *zilde*. Note that *HasCheckBox* may only be set when the object is created.

3.135 HasClearButton Property

Applies To: *ButtonEdit*, *Combo*, *ComboEx*, *Edit*

Description

Specifies whether or not a



button is displayed in the right-hand end of an edit box. Clicking this button clears the text from the field.

Note

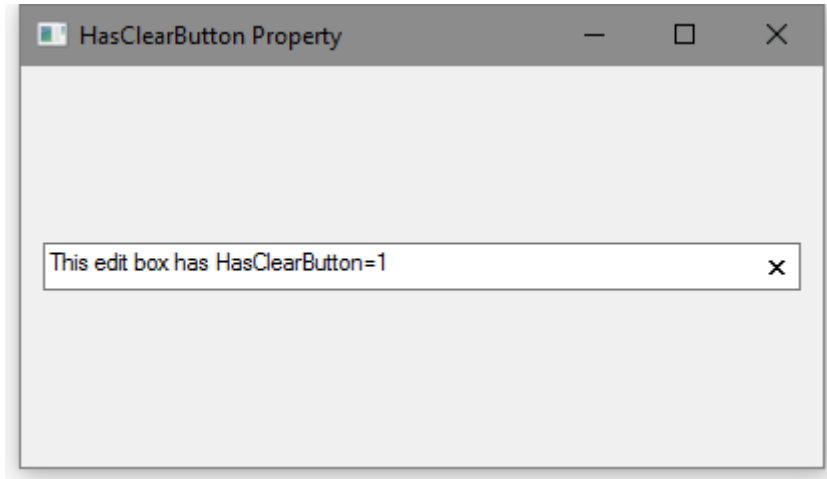
This feature only applies if *Native Look and Feel* is enabled.

HasClearButton is Boolean. 1 means that a



button will be displayed; 0 (the default) means that the button will not be shown. It may only be specified when the object is created. If you subsequently attempt to change the value of `HasClearButton`, the operation will fail with `NONCE ERROR`.

`HasClearButton` is only effective for Edit objects with Style Single; it is silently ignored for other Styles of Edit objects.



3.136 HasEdit Property

Applies To: *BrowseBox*

Description

Specifies whether or not a *BrowseBox* has an edit field.

`HasEdit` is a single number with the value 0 (the default) or 1. If `HasEdit` is 1, the user may type in the name of a folder or other resource that is the target of the *BrowseBox*. If `HasEdit` is 0, the user must browse to it.

3.137 HasHelp Property

Applies To: *PropertyPage*, *PropertySheet*

Description

The `HasHelp` property is a Boolean value. For a *PropertySheet*, it determines whether or not the *PropertySheet* has a Help button. For a *PropertyPage*, `HasHelp` determines whether or not the Help button is active when the *PropertyPage* is the current page. If

the `HasHelp` property of a *PropertyPage* is 0, the Help button on the parent *PropertySheet* will be temporarily disabled when that *PropertyPage* is displayed.

3.138 HasLines Property

Applies To: *TreeView*

Description

The `HasLines` property specifies whether or not tree lines are drawn in a *TreeView* object. It is a single integer with the value 0, 1 or 2:

0	No tree lines
1	Tree lines are drawn at all levels except the top level
2	Tree lines are drawn at all levels

The user can cause a parent item to expand or collapse by clicking on its corresponding tree line.

3.139 HasTicks Property

Applies To: *TrackBar*

Description

The `HasTicks` property specifies whether or not tick marks are drawn in a *TrackBar* object. It is Boolean value with a default value of 0.

The position of the tick marks in the *TrackBar* is determined by the *TickAlign* property.

3.140 HasToday Property

Applies To: *Calendar*, *DateTimePicker*

Description

The `HasToday` property specifies whether or not the *Today* date is displayed in the bottom left corner of a *Calendar* object or in the drop-down calendar of a *DateTimePicker*.

HasToday is a single number with the value 0 (the date is *not* shown) or 1 (the date is shown); the default is 1.

See also *CircleToday* property.

3.141 Header Property

Applies To: *ListView*

Description

The Header property is Boolean and specifies whether or not a *ListView* object displays column titles. Its default value is 1. Header applies only if the *View* property is 'Report'. The column titles are defined by the *ColTitles* property and their alignment by the *ColTitleAlign* property.

Note that Header may only be set by QWC and may not subsequently be changed.

3.142 HeaderImageIndex Property

Applies To: *ListView*

Description

The HeaderImageIndex property is an integer vector that specifies the images to be displayed alongside each column heading in a *ListView* object in Report View. Each positive element of HeaderImageIndex specifies an index into the *ImageList* object specified by the *HeaderImageList* property. The special values -1 and -2 specify the standard *Sort Up* and *Sort Down* images respectively.

3.143 HeaderImageList Property

Applies To: *ListView*

Description

The HeaderImageList property specifies the name of or ref to an *ImageList* object that contains images to be displayed alongside each column heading in a *ListView* object in Report View.

3.144 HelpButton Property

Applies To: *Form, PropertySheet, SubForm*

Description

This is a Boolean property that specifies whether or not a Question (?) button appears in the title bar of a Form or SubForm. However, this does not apply if the Form has a maximise or minimise button which both take precedence. The user may obtain help by clicking on the Question (?) button and then on a control in the Form. It is up to you to provide the help by responding to the *Help* event on the control. The default value of HelpButton is 0.

3.145 HelpFile Property

Applies To: *ActiveXControl, OCXClass, OLEClient*

Description

The HelpFile property is a read-only property that reports the pathname of a Windows help file associated with a particular COM object.

For an *OCXClass* or *OLEClient* object, the HelpFile property is read-only.

3.146 HighlightHeaders Property

Applies To: *Grid*

Description

The HighlightHeaders property is a Boolean value (default 1) that specifies whether or not the appropriate row and column titles in a *Grid* are highlighted corresponding to the currently selected block of cells.

3.147 Hint Property

Applies To

Description

The *Hint* property is a character vector that specifies a *help* message that is to be displayed when the user positions the mouse pointer over the object. The *Hint* is displayed in the object specified by its *HintObj* property. A *StatusField* is often used for this purpose.

3.148 HintObj Property

Applies To

Description

The *HintObj* property is a character vector or ref that specifies the name of, or a ref to, an object in which the "help" message defined by the *Hint* property is to be displayed. This message is displayed when the user positions the mouse pointer over the object. The *Hint* is displayed by automatically setting the *Caption* or *Text* property of the object named by *HintObj*.

The following types of object can therefore be used to display Hints: *Button*, *Edit*, *Combo*, *Group*, *Form*, *Label*, *Menu*, *MenuItem*, *StatusField*, *SubForm* and *Text*. For a *StatusField* that has both *Caption* and *Text* properties, the text property is used for displaying hints.

When the user moves the mouse pointer away from the object, the *Caption* or *Text* property of the object specified by *HintObj* is reset to an empty vector.

Note that if *HintObj* is empty, its value is inherited from its parent. Thus setting *HintObj* on a *Form* defines the default location for displaying *Hints* for all the controls in that *Form*. Setting *HintObj* on *Root* defines the default location for hints for the entire application.

3.149 HotSpot Property

Applies To: *Cursor*

Description

This property specifies the point within a *Cursor* object that registers the cursor's position over another object. The mouse position, which is reported by various events, is actually the position of the cursor's *HotSpot* over the object in question.

HotSpot is a 2-element numeric vector that specifies the y-position and x-position of the hotspot within the cursor. A value of (0 0) specifies the top-left corner of the cursor;

(31 31) specifies the bottom right corner of the cursor. The default value of `HotSpot` is (16 16).

3.150 HotTrack Property

Applies To: *TabControl*

Description

The `HotTrack` property specifies whether or not the tabs or buttons in a *TabControl* object (which are represented by *TabButton* objects), are automatically highlighted by the mouse pointer.

`HotTrack` is a single number with the value 0 (no highlighting) or 1. The default is 0.

If `HotTrack` is 1 and the `Style` property of the *TabControl* is ' `Tabs` ' or ' `Buttons` ', the text defined by the `Caption` property of the *TabButton* is highlighted when the mouse pointer is placed over the tab or button. If `Style` is ' `FlatButtons` ', the button is highlighted by being *raised*.

The value of `HotTrack` is effective only when the object is created with `□wc`.

3.151 IconObj Property

Applies To

Description

This property is used to specify a *large* and *small* icon for a *Form* or *SubForm*, or for the *Root* object which represents your application as a whole. Its value is either a single ref or character scalar or vector containing the name of, or ref to, an *Icon* object, or a 2-element vector of character vectors or refs that specifies 2 *Icon* objects.

If empty (the default value), the standard "Dyalog APL GUI" icon is used.

The large and small icons are supplied to the Operating System which uses them as and when is appropriate. Normally, the large icon is of size 32x32 and the small icon is 16x16. If you specify an icon of a different size, the Operating System will scale it as appropriate.

The icon associated with a *Form* is displayed when the *Form* is minimised. The icon associated with ' . ' is displayed when the user presses Alt+Tab to toggle between

applications. In both cases, the text shown underneath or alongside the icon is defined by the *Caption* property.

3.152 ImageCount Property

Applies To: *ImageList*

Description

The ImageCount property is a read-only property that reports the number of images in an *ImageList* object. It is an integer scalar.

3.153 ImageIndex Property

Applies To

Description

For a *ComboEx*, *ListView* or *TreeView*, the ImageIndex property maps bitmapped images in an *ImageList* to items. ImageIndex is an integer vector whose length is the same as the number of items in the object. See also *SellImageIndex*

For a *CoolBand*, *MenuItem*, *TabButton* or *ToolButton*, ImageIndex specifies the picture to be displayed in the object. In these cases, ImageIndex is a single integer value.

ImageIndex is ☐IO dependent.

3.154 ImageListObj Property

Applies To

Description

The ImageListObj property is a simple character vector or a ref, or a vector of character vectors or refs that specifies *ImageList* objects that are associated with an object.

For a *ComboEx* or *TreeView* object, the ImageListObj property specifies the name of, or ref to, a single *ImageList* object that contains a set of images to be displayed alongside its Items. The image(s) displayed by a particular item in its normal (unselected) and selected states are specified by the corresponding element of the *ImageIndex* and *SellImageIndex* properties respectively.

For *CoolBar*, *Menu*, and *TabControl* objects, the *ImageListObj* property specifies the name of, or ref to, a single *ImageList* object that contains a set of images for its *CoolBand*, *MenuItem*, and *TabButton* children respectively.

For a *ToolControl*, *ImageListObj* may specify up to three *ImageList* objects that correspond to the three different states, normal, highlighted (hot) and inactive, of its *ToolButton* children.

In all these cases, individual images are mapped to the child objects by their *ImageIndex* property.

For a *ListView* either one or two *ImageList* objects may be specified. The first *ImageList* contains the *large icon* set of images. the second contains the *small icon* set. The set that is used is determined by the value of the *View* property. The mapping between the set of images in the *ImageList* and items in the object is determined by the *ImageIndex* property.

3.155 Indents Property

Applies To: *ComboEx*

Description

Specifies the amount by which items in a *ComboEx* object are indented.

Indents may be an integer scalar or a vector with the same number of elements as there are items in the *ComboEx*. Its default value is 0.

The unit of indenting is 10 pixels. For example, if there are 3 items and Indents is (0 1 2), the items will be indented by 0, 10 and 20 pixels respectively.

3.156 Index Property

Applies To

Description

For a *List* and a *Combo* with *Style* 'Simple', this property specifies the position of the data in the list box as a positive integer value. If *Index* has the value "n", it means that the "nth" item in *Items* is displayed on the top line in the list box. The value of *Index* is dependent upon the value of $\square IO$. Note that *Index* for a *Combo* or *List* cannot be set using *Dyalog APL Language: Wc*. The value of *Index* in a *Combo* with a drop-down list box (*Style* 'Drop' or 'DropEdit') is always equal to $\square IO$.

For a *Grid*, *Index* is a 2-element vector that specifies the row and column number of the cell that is currently in the top left corner of the *Grid*.

For a *TreeView*, *Index* is a positive integer value that specifies which item appears at the top of the *TreeView* window.

For a *FileBox*, the *Index* property determines which of the *Filters* is initially selected.

For a *CoolBand*, the *Index* property specifies the position of the *CoolBand* within its parent *CoolBar*, relative to the other *CoolBands* in the *CoolBar*.

The value of *Index* is dependent on $\square IO$, and its default value is equal to $\square IO$.

3.157 Input Property

Applies To: *Grid*

Description

This property specifies objects to be associated with cells in a *Grid*.

These objects may be of type *Button*, *ColorButton*, *Combo*, *Edit*, *Label*, *Spinner* or *TrackBar*. In addition, the *Input* property may specify instances of *OCXClass* objects (ActiveX controls) and *NetType* objects (.NET classes).

The *Input* property is either a single ref or a simple character scalar or vector, or a vector of character vectors or refs. If it specifies a single object, this will be associated with *all* of the cells in the *Grid*. If it specifies a set of objects, these are mapped to individual cells through the *CellTypes* property.

When a cell becomes the current cell, its value (defined by the appropriate element of the *Values* property) defines the value of a *corresponding property* of the associated object. The property that corresponds to the value in the cell, depends upon the *Type* of the associated object as shown in the following table:

Associate Object Type	Corresponding Property
<i>Label</i>	<i>Text</i>
<i>Edit</i>	<i>Value</i>
<i>Combo</i>	<i>Text</i>
<i>Button</i> (<i>Style Push</i>)	<i>Caption</i>
<i>Button</i> (<i>Style Radio or Check</i>)	<i>State</i>
<i>ColorButton</i>	<i>CurrentColor</i>
<i>Spinner</i>	<i>Value</i>
<i>TrackBar</i>	<i>Thumb</i>
<i>OCXClass</i>	Specified by <i>InputProperties</i>
<i>NetType</i>	Specified by <i>InputProperties</i>

In effect, the user inputs a new value into the current cell by changing the corresponding property of its associated object. An associated object may be a *fixed* object that is *external* to the *Grid* or a *floating* object that moves automatically from cell to cell. The latter is achieved by creating the associated object as a *child* of the *Grid*.

If the associated object is an *Edit* or *Combo*, the user may change the *Text* property of the object by typing, or, in the case of a *Combo*, by selecting an item from a list. The new value of the *Text* property is then used to update the value in the cell (defined by the *Values* property of the *Grid*) when the user moves on. If the associated object is a radio button, the value in the cell (0 or 1) is reflected by the *State* property of the *Button*. The user may click the *Button* on and off, changing its *State* and thereby the corresponding value in the cell.

If the associated object is a *ColorButton*, the corresponding elements of the *Values* property contain 3-element integer vectors which specify the RGB colour values.

If the associated object is an instance of an *OCXClass* object (ActiveX control) or a *NetType* object (.NET class), the *Grid* uses the *default* property of the external object if it has one. Alternatively, the *InputProperties* property is used to specify which property (or properties) of the external object are to be mapped to elements of *Values*. If more than one property is specified, elements of *Values* are vectors.

If there is no object associated with a cell, or if its associated object is a *Label* or a *Button* with *Style Push*, the cell is protected and may not be changed by the user. When

the current cell is thus protected, the corresponding row and column titles are not indented as they are when the cell may be edited.

If the associated object is a numeric *Edit* or *Label* (*FieldType* Numeric, LongNumeric, Currency, Date, LongDate or Time) the contents of the cell are formatted accordingly, even when it is not the current cell. Thus a cell associated with a *Label* with *FieldType* Date, always displays as a date.

If the associated object is a *Combo* or *Button*, the appearance of a non-current cell is defined by the corresponding element of the *ShowInput* property.

The following example illustrates the use of different types of object specified by the *Input* and *ShowInput* properties:

Different Input Objects						
Surname	Job Title		Region		Salary	Permanent
Brown	Manager	↓	South	↓	\$64000.00	☒
Jones	Project Leader	↓	South	↓	\$43250.00	☒
Green	Consultant	↓	South	↓	\$45000.00	☐
Black	Programmer	↓	East	↓	\$30000.00	☒
White	Assistant	↓	Central	↓	\$40000.00	☐

3.158 InputMode Property

Applies To: *Grid*

Description

This property determines editing behaviour and the action of the cursor movement keys when the user changes the contents of a Grid using a *floating Edit* or *DropEdit Combo* control.

InputMode is a character vector with one of the following values:

'Scroll'	The cursor keys move around the Grid; the user may switch to <i>InCell</i> mode
'InCell'	The cursor keys move within the Input object; the mode reverts to <i>Scroll</i> when the user selects a new cell
'AlwaysScroll'	The cursor keys move around the Grid; the user may not switch to <i>InCell</i> mode
'AlwaysInCell'	The cursor keys move within the Input object, even when the user moves to a new cell
'AutoEdit'	See below

By default, the input mode is *Scroll*. In this mode, cursor movement keys are actioned by the Grid itself and used to move from cell to cell. The user may switch to *InCell* mode by double-clicking or by pressing the key defined by *InputModeKey* (the default is "F2").

In *InCell* mode, all cursor movement keys are actioned by the Input object and typically move the cursor around *within* the Input object and do not switch between cells. When the user switches to a different cell, *InputMode* reverts to *Scroll* mode

If *InputMode* is *AlwaysScroll* or *AlwaysInCell*, the user remains permanently in either *Scroll* or *InCell* mode respectively.

If *InputMode* is 'AutoEdit', the behaviour of a cell that contains a floating Input field is as follows:

When the user enters the cell, the contents are selected (and highlighted). At this stage, the cursor movement keys move to an adjacent cell. If the user presses a (valid) data key, that character replaces the current contents of the cell.

If the user presses F2 (or the key defined by the *InputModeKey* property), the data is de-selected and unhighlighted and the cursor is placed at the rightmost end of the data.

In either case, the left and right cursor keys now move the cursor within the current data string, but skip to the adjacent cell from the beginning or end of the data. This behaviour differs from *InCell* mode in which the cursor movement keys *stick* at the end of the data.

3.159 InputModeKey Property

Applies To: *Grid*

Description

This property defines the keystroke used to switch from Scroll mode to InCell mode in a *Grid*. It applies only where the *Grid* has a *floatingEdit* control. See the description of the *InputMode* property for further details.

The InputModeKey property is specified (in the same way as the *Accelerator* property) as a 2-element vector of integer values containing the key number and shift state respectively. Its default is (113 0) which is "F2".

As an example, if you wanted to use Ctrl+Shift+a to switch modes, you would set InputModeKey to (65 3). 65 is the keynumber for "a" and 3 means Shift (1) + Ctrl (2).

3.160 InputProperties Property

When an ActiveX Control or .NET Class is used as a child of the *Grid*, InputProperties is used to specify how the value in each *Grid* cell corresponds to the value of one or more properties of the child object.

For example, suppose there is a third-party ActiveX Control that displays a playing card. The control has two properties named Suit and Value that specify the suit (1=clubs, 2=diamonds, 3=hearts, 4=spades) and card value (1="Ace", 2="2", ...11="Jack",...) respectively. To display these cards in a *Grid*, the InputProperties property may be set to ('Suit' 'Value') and each element of the *Values* property must be a 2-element integer vector specifying the suit and value of the corresponding card.

```
'CARDS' WC OCXClass '...'
'F' WC Form
'F.G' WC Grid
'F.G.card' WC CARDS
F.G.Input←'F.G.card'
F.G.InputProperties←'Suit' 'Value'
F.G.Values←{4 13}
```

If InputProperties is not specified, the *default* property of the ActiveX Control or .NET Class is used.

3.161 InstanceMode Property

Applies To: *OLEClient*

Description

The InstanceMode property specifies how APL attempts to connect the *OLEClient* to an OLE Server.

InstanceMode is a character vector that may be 'ExistingFirst' (the default), 'ExistingOnly' or 'New'. Its value is effective only when the object is created with `⎕WC`. Changing InstanceMode with `⎕WS` has no effect.

If InstanceMode is 'ExistingFirst', APL attempts first to connect to a running instance of the OLE Server. If there is no running instance, it starts the OLE server to obtain a new object.

If InstanceMode is 'ExistingOnly', APL attempts to connect to a running instance of the object. If there is no running instance, it fails with a `DOMAIN ERROR`.

Note that in either case, if there is more than one instance running, there is no way to predict to which instance APL will be connected.

If InstanceMode is 'New', APL attempts to start the OLE Server to obtain a new object, whether or not the OLE Server is already running. However, if the OLE Server has registered itself as a single instance object and is already running, APL will be connected to it, and a second instance of the Server will not in fact be started.

3.162 InterceptedURLs Property

Applies To: *HTMLRenderer*

Description

The InterceptedURLs property is a 2-column matrix that specifies whether the *HTMLRenderer* will attempt to satisfy a request for a resource from the workspace or, via the CEF, from the internet. If directed to the workspace, the request will trigger an *HTTPRequest* event if the protocol is `http`, or a *WebSocketUpgrade* event if the protocol is `ws`.

The first column is a wild-carded character scalar or vector containing a pattern to match. The second column is numeric indicating whether or not the *HTMLRenderer*

should trigger an event as shown in the table below. InterceptedURLs may contain any number of rows.

Value	Meaning
0	Ignore request; pass to CEF for fulfilment
1	Trigger an <i>HTTPRequest</i> or <i>WebSocketUpgrade</i> (automatic) in the workspace
2	Trigger an <i>HTTPRequest</i> or <i>WebSocketUpgrade</i> (manual) in the workspace

If the requested url is a relative rather than an absolute URL, it is prepended by the string `http://dyalog_root/`. So, for example, if the *HTML* property contains :

```
<link rel="stylesheet" href="style.css">
<script src="app.js"></script>
```

the *HTMLRenderer* will request `http://dyalog_root/style.css` and `http://dyalog_root/app.js` respectively.

When the value of *InterceptedURLs* is its default (`0 2p''`) it is treated as if it were set to (`1 2p'*://dyalog_root/*' 1`). So by default, requests for a relative URL will fire an event in the workspace while absolute URL will be directed by the CEF to the internet.

Note that if code in the page creates a web socket intended for internal use, with anything other than `dyalog_root` as the URL, the URL must match a pattern in *InterceptedURLs* with 1 in the second column. The following example does not require a matching pattern in *InterceptedURLs*.

```
// Create a new WebSocket.
window.socket = new WebSocket('ws://dyalog_root/');
```

EXAMPLES

The following will trigger an event for all requested URLs.

```
InterceptedURLs ← 1 2p'*' 1
```

The following will attempt to retrieve from the net URLs containing `'.dyalog.com'` and trigger an *HTTPRequest* event for all other requested URLs.

```
InterceptedURLs ← 2 2p'*..dyalog.com*' 0 '*' 1
```

3.163 Interval Property

Applies To: *ProgressBar*, *Timer*

Description

Interval is an integer value specified in milliseconds and has a default of 1000.

For a *Timer* object, the Interval property specifies the frequency with which it generates *Timer* events. Setting Interval to 0 disables *Timer* events.

For a *ProgressBar* with *ProgressStyle* 'Marquee', Interval specifies the frequency with which the animated bar is updated. Setting Interval to -1 stops the animation.

3.164 IsLoading Property

Applies To: *HTMLRenderer*

Description

IsLoading is a Boolean property whose value is 1 if the browser is currently loading, or 0 when the frame content is completely loaded.

3.165 Italic Property

Applies To: *Font*

Description

This property specifies whether or not a font represented by a *Font* object is italicised or not. It is either 0 (normal) or 1 (italic). There is no default; the value of this property reflects the characteristic of the selected font.

3.166 ItemGroupMetrics Property

Applies To: *ListView*

Description

This property is used to specify colours and spacing elements for a *ListView* that is displaying its Items in groupings (see *ItemGroups*).

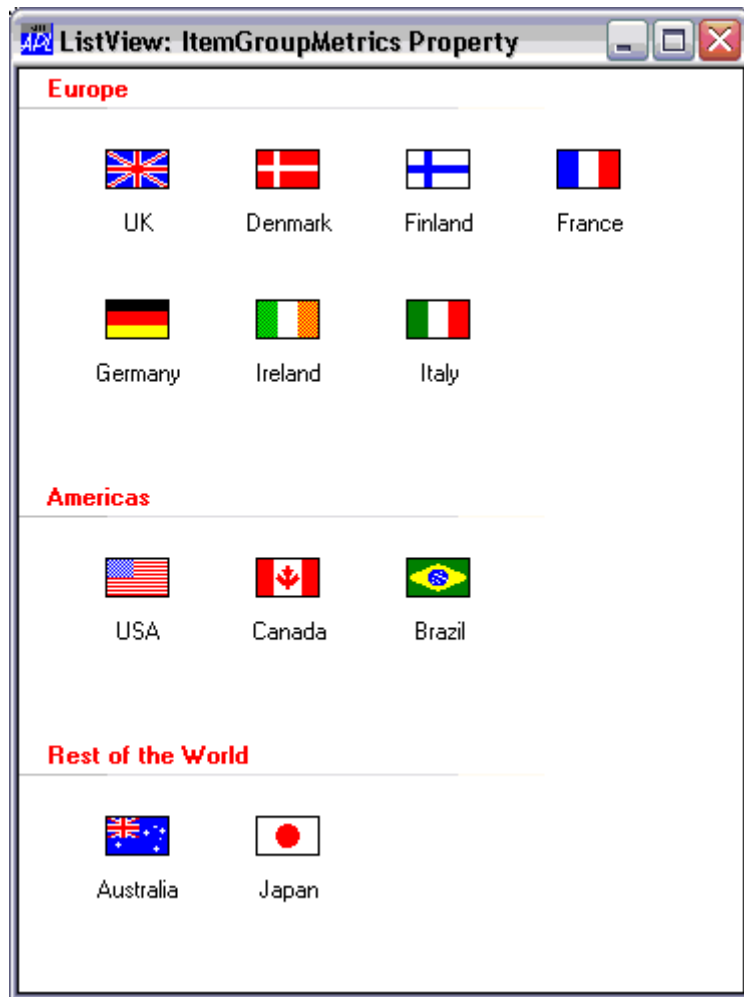
****Note that this feature only applies if Native Look and Feel is enabled.****

ItemGroupMetrics is a 3-item nested vector as follows:

[1]	Text Colours	2-element vector of 3 element RGB values that specifies the colour of the group caption and group footer respectively.
[2]	Spacing	4-element integer vector that specifies the top, left, bottom and right spacing around each grouping in pixels
[3]	Border	Colours4-element vector of 3 element RGB values that specifies the colours for the top, left, bottom and right borders (not yet implemented).

The following expression, coupled with the code shown in the SetGroups example, causes the items to be displayed as shown below.

```
F.L.ItemGroupMetrics[1 2]←(2p<255 0 0)(10 100 0 10)
```



3.167 ItemGroups Property

Applies To: *ListView*

Description

This property specifies item groupings for a *ListView* object.

Note

This feature only applies if *Native Look and Feel* is enabled.

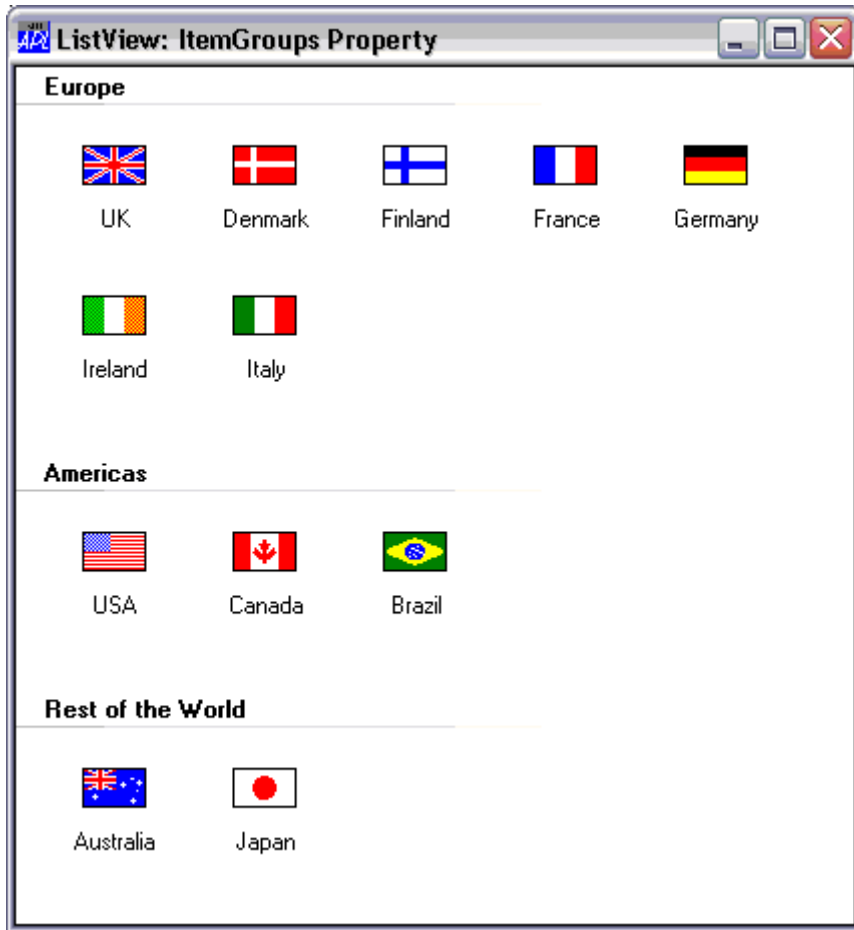
ItemGroups is a nested scalar or nested vector each of whose elements specifies a grouping. Each grouping is a 5-element vector as follows:

[1]	Group caption	character vector
[2]	Item index	Vector of indices to the Items property that specifies which Items are in this grouping.
[3]	Caption alignment	an integer: 1 = left aligned caption (the default) 2 = centre aligned caption 4 = right-aligned caption
[4]	State	Integer (not yet implemented)
[5]	Footer text	character vector (not yet implemented)

Note that State and Footer text are not yet implemented by Windows.

For example, the following expressions executed in the WTUTOR95 workspace will result in the display shown below.

```
'F'[]WC'Form' 'ListView Object'
II←ipCOUNTRIES
'F.L'[]WC'ListView'COUNTRIES(0 0)(100 100)
      ('ImageList' 'F.I1')
      ('ImageIndex' (ipCOUNTRIES))
'F.I1'[]WC'ImageList'('Size' 32 32)
(←'F.I1.')[]WC''(←'Icon' ''),''↓↑FLAGBITS FLAGCMAP FLAGMASK
GROUPS←←'Europe'(11 4 5 6 7 8 9)
GROUPS,←←'Americas'(12 3 2)
GROUPS,←←'Rest of the World'(1 10)
F.L.ItemGroups←GROUPS
```



You can control the appearance of the groupings using the *ItemGroupMetrics* property.

3.168 Items Property

Applies To: *Combo*, *ComboEx*, *List*, *ListView*, *Spinner*, *TreeView*

Description

This property specifies the list of items from which the user may choose.

The value of *Items* is a text array. It is normally specified as a vector of character vectors each of which represents an item. For a *Combo*, *ComboEx*, or *List* or *Spinner*, *Items* may also be a matrix whose rows specify items. If a character scalar or simple vector is specified, it is treated as a single item.

An empty character vector is treated the same as a vector of blanks, and represents one item.

A zero-length vector of vectors or an empty matrix represents 0 items. The default value for `Items` is an empty matrix.

`WG 'Items'` returns an array of the same structure as was assigned by *Dyalog APL Language*: `Wc` or *Dyalog APL Language*: `Ws`.

3.169 Justify Property

Applies To: *Button, ButtonEdit, Edit, Label, Spinner, TabControl*

Description

This property determines the manner in which text is justified within the object. It is a character vector that may take the value `'Left'` (the default), `'Centre'` or `'Right'`. The keyword `'Centre'` may also be spelled `'Center'`.

When applied to an *Edit* object with *Style* `'Multi'`, a value of `'Centre'` or `'Right'` forces word-wrapping and disables horizontal scrolling. Note that *Justify* only applies to a multi-line edit field. If you specify a value for *Justify* in a 1-line edit field (*Style* `'Single'`), it will be ignored.

For a *TabControl*, *Justify* may be `'Right'` (which is the default) or `'None'` or empty.

If *Justify* is `'Right'`, the *TabControl* increases the width of each tab, if necessary, so that each row of tabs fills the entire width of the tab control. Otherwise, if *Justify* is empty or `'None'`, the rows are ragged.

With the exception of *Label* and *TabControl* objects, *Justify* may only be specified when the object is created using `WG`.

3.170 KeepBits Property

Applies To: *Bitmap, Cursor, Icon*

Description

This property is be used to control the way that *Bitmap*, *Cursor* and *Icon* objects are stored in the workspace.

When you create a *Bitmap*, *Icon* or *Cursor* using `␣WG`, APL asks Windows to allocate a corresponding bitmap, icon or cursor *resource*. This resource is allocated in Windows memory. If APL were to hold the values of the *image properties* (*CBits*, *Bits* and *CMap* for a *Bitmap*; *Bits*, *CMap* and *Mask* for *Cursor* and *Icon* objects) internally in the workspace, this data would be duplicated. For large bitmaps this would have a serious impact on memory utilisation and may affect performance. The *KeepBits* property is provided to allow you to control whether or not APL retains the values of the *image properties* in the workspace, so that you can choose a strategy to suit your configuration and requirements. *KeepBits* may take the value 0 or 1.

If *KeepBits* is 0 the values of the *image properties* are **not** stored internally in your workspace. If you save a workspace containing a *Bitmap*, *Cursor* or *Icon* object, the corresponding Windows resource is automatically re-allocated when the workspace is loaded by referring to the associated file. This is the file whose full pathname is defined by the value of the object's *File* property. It follows that if you adopt this strategy, you must ensure that the *File* property is set correctly. If APL cannot find the file when the workspace is `)LOADED`, it cannot re-create the object, and you will get a `VALUE ERROR` when you subsequently refer to it. A further consideration is the effect on `␣WG`. If *KeepBits* is 0, and you execute `␣WG 'CBits'` or `'Bits'` or `'CMap'` or `'Mask'`, APL obtains these values by requesting the data from Windows.

If *KeepBits* is set to 1, the contents of the *image properties* are stored in the workspace, thus duplicating the information which is held by Windows itself. If you save a workspace containing a *Bitmap*, *Cursor* or *Icon* the corresponding Windows resource is automatically re-allocated from the *image properties* when the workspace is loaded. The value of the *File* property is ignored. When you execute `␣WG 'CBits'` or `'Bits'` or `'CMap'` or `'Mask'`, APL generates the result directly from the stored values held (internally) in the workspace.

3.171 KeepOnClose Property

Applies To

Description

This property is either 0 or 1 and determines how the object is treated when its parent Form (or, in the case of a Form, the Form itself) is closed by the user, receives a Close event from `␣NQ`, or when Close is called as a method.

If *KeepOnClose* is 1 (for the object itself **and** for all its parents) when its parent Form is closed, the object changes from being a GUI object to a pure namespace. For example, the Type of a Button will change from `'Button'` to `'Namespace'`. Effectively, the GUI

component of the object is discarded but its Namespace component (and any variables, functions, operators and other namespaces that it contains) remains intact. Monadic `□wc` may subsequently be used to re-attach the GUI component to the object. All child GUI objects are treated in the same way.

Note that the default value of `KeepOnClose` depends upon the way in which a GUI object was created with `□wc`. If a GUI object is created by dyadic `□wc`, `KeepOnClose` defaults to 0. If a GUI object is attached by monadic `□wc`, its `KeepOnClose` property defaults to 1.

Note

The use of `KeepOnClose` is deprecated; Dyalog may at some point remove this Property or disable its functionality.

3.172 LStyle Property

Applies To: *Circle, Ellipse, Locator, Poly, Rect*

Description

This property determines the type of line used to draw a graphics object. It takes one of the following integer values, or, if the object contains more than one component, a vector of such values.

0	solid line
1	dashed line
2	dotted line
3	dash dotted line
4	dash dot dotted line
5	null line (invisible)

If `LStyle` is in the range 1-4, the gaps between the dashes and dots are drawn using the colour specified by `BCol`, or are left undrawn (that is, transparent) if `BCol` is not defined.

In versions of Dyalog prior to v13.2 revision 19489, if *LWidth* specified a line width greater than 1 pixel, a solid line was drawn in the colour specified by the *FCol* Property, regardless of the value of *LStyle*. From that revision onwards, if the value of *LWidth* is greater than 1 then the value of *LStyle* is honoured, but only the *FCol* of the line is honoured - the *BCol* is still ignored.

3.173 *LWidth* Property

Applies To: *Circle, Ellipse, Poly, Rect*

Description

This property determines the width of line used to draw a graphics object. A positive value specifies the line width in pixels. A negative value specifies line width in units of the co-ordinate system defined for the object in the x direction. If the object contains more than one component, *LWidth* may be a vector.

In versions of Dyalog prior to v13.2 revision 19489, if *LWidth* specified a line width greater than 1 pixel, a solid line was drawn in the colour specified by the *FCol* Property, regardless of the value of *LStyle*. From that revision onwards, if the value of *LWidth* is greater than 1 then the value of *LStyle* is honoured, but only the *FCol* of the line is honoured - the *BCol* is still ignored.

3.174 *LastError* Property

Applies To: *ActiveXControl, OLEClient, OLEServer, Root*

Description

The *LastError* property provides information about the most recent error reported by OLE. You may use this property to report an error from an *OLEServer* or *ActiveXControl* to a host application.

3.175 *LateBind* Property

Applies To: *OLEClient, OLEServer*

Description

LateBind is a Boolean property that determines whether or not the Type Information is read when an *OLEClient* object is instantiated, or when an *OLEServer* references a COM

object. It must be set when the *OLEClient* or *OLEServer* object is created and cannot subsequently be changed.

If *LateBind* is 0 (the default) the Type information is obtained, in its entirety, when the *OLEClient* object is instantiated or when an *OLEServer* processes a reference to an external COM object.

If *LateBind* is 1, APL postpones the calls to obtain Type Information until a particular property, method or event of the COM object is referenced; and then only for that particular member.

3.176 LicenseKey Property

Applies To: *OCXClass*

Description

The *LicenseKey* property is a character string that contains the license key for an ActiveX control.

If an ActiveX control requires a license key, it must be specified by an application when it creates an instance of the control. Typically, the license key is *required* only by the run-time version of an ActiveX control, and is *made available* to an application by the development version of the control.

3.177 Limits Property

Applies To: *ProgressBar*, *Scroll*, *Spinner*, *TrackBar*, *UpDown*

Description

This property is a 2-element vector that specifies the minimum and maximum values of an object. The values must be in the range (-2×31) to $(-1 + 2 \times 31)$ inclusive.

3.178 LocalAddr Property

Applies To: *TCPSocket*

Description

The *LocalAddr* property is a character vector that specifies the IP address of your computer. Its default value is '0.0.0.0'.

Unless your computer has more than one network adapter each identified by a different IP address, you do not need to specify `LocalAddr`. However, in this case you may use *either* `LocalAddr` or `LocalAddrName` to identify the adapter. If you specify both properties, the value of `LocalAddrName` will be ignored.

Note that you may also set the value of `LocalAddr` to an empty character vector. In this case, the value returned by `WG` will be `'0.0.0.0'`.

`LocalAddr` may only be specified in the `WG` statement that creates the `TCP Socket` and may not subsequently be changed using `WS`.

3.179 LocalAddrName Property

Applies To: `TCP Socket`

Description

The `LocalAddrName` property is a character vector that specifies the host name of your computer. It may be useful when you have more than one network adapter (perhaps an Ethernet adapter and a token ring adapter) and you wish to avoid hard-coding the IP address.

Note that you may use *either* `LocalAddr` or `LocalAddrName` to identify the local computer. If you specify both properties, the value of `LocalAddrName` will be ignored.

`LocalAddrName` may only be specified by a server `TCP Socket`. Furthermore, it must be specified in the `WG` statement that creates the `TCP Socket` object and it may not subsequently be changed using `WS`.

When the specified host name has been resolved to an IP address, the `TCP Socket` will generate a `TCPGotAddr` event and update the value of `LocalAddr` accordingly.

For a client `TCP Socket`, you may not specify `LocalAddrName` and `WG` returns an empty character vector.

3.180 LocalPort Property

Applies To: `TCP Socket`

Description

The `LocalPort` property is a scalar integer in the range 1-65536 that identifies the *port number* associated with a `TCP Socket` object.

Note that you may use *either* `LocalPort` or `LocalPortName` to identify the service. The use of `LocalPortName` is slightly slower but it avoids hard-coding the port number in your program and is generally more flexible. If you specify both properties, the value of `LocalPortName` will be ignored.

`LocalPort` may be specified only by the process that is initiating the connection (the server) and must be set by the `□WC` statement that creates the `TCPSocket`. `LocalPort` may not subsequently be changed using `□WS`.

If you specify a value of 0, the system will assign an available port number. For example:

```
'S1' □WC'TCPSocket' ( 'LocalPort' 0)
S1.LocalPort
4047
```

For a process that is completing a connection, `LocalPort` is allocated by the system and is effectively read-only.

3.181 LocalPortName Property

Applies To: `TCPSocket`

Description

The `LocalPortName` property is a character vector that specifies the port name of the local service that you wish to offer as a server.

Note that you may use *either* `LocalPort` or `LocalPortName` to identify the service. The use of `LocalPortName` is slightly slower but it avoids hard-coding the port number in your program and is generally more flexible. If you specify both properties, the value of `LocalPortName` will be ignored.

`LocalPortName` may be specified only by the process that is initiating the connection (the server) and must be set by the `□WC` statement that creates the `TCPSocket`. `LocalPortName` may not subsequently be changed using `□WS`.

When the specified port name has been resolved to a port number, the `TCPSocket` will generate a `TCPGotPort` event and update the value of `LocalPort` accordingly.

For a client `TCPSocket`, you may not specify `LocalPortName` and `□WC` returns an empty character vector.

3.182 Locale Property

Applies To: *OLEClient*

Description

The Locale property specifies the language in which the OLE server, attached to an OLEClient, exposes its methods (functions) and properties (variables).

When you create an OLEClient object, Dyalog APL/W requests the default Type Library associated with the OLE server that you specify. Many OLE servers, such as Excel.Application, provide different names for the methods and properties they expose for different languages. Without Locale, it would be difficult to write an OLE client application that could run in different countries, as the names of the functions and variables may be unpredictable.

Locale is an integer; for example, the value 9 specifies English and the value 12 specifies French.

Locale may **only** be specified by the `⎕WC` statement that is used to create the OLEClient; it may not subsequently be changed using `⎕WS`. A table of commonly used Locale values is given below.

Note that Dyalog cannot guarantee that you will actually be given the Locale you specify. This is a function of your specific installation and the OLE server in question. However, Dyalog believes that for Microsoft products, it is a fairly safe bet that the US/English interface will be available in most countries.

Language	Locale
Neutral	0
Danish	6
Dutch	19
English	9
Finnish	11
French	12
German	7
Italian	16
Norwegian	20
Portuguese	22
Russian	25
Spanish	10
Swedish	29

3.183 MDIActive Property

Applies To: *MDIClient*

Description

This property contains the name of the *SubForm* owned by the *MDIClient* that is currently active. Only one *SubForm* may be active at a time. You can switch between *SubForms* in an MDI application under program control by setting this property with *Dyalog APL Language: Ws*. You can also do this by generating an *MDIActivate* event.

See also *MDIActiveObject* property.

3.184 MDIActiveObject Property

Applies To: *MDIClient*

Description

This property contains a ref to the *SubForm* owned by the *MDIClient* that is currently active.

See also *MDIActive* property.

3.185 MDIMenu Property

Applies To: *MenuBar*

Description

This property specifies the name of, or ref to, the *Menu* object that is nominated as the "Window" menu in an MDI application. If such a menu is defined, the *Captions* of all the child *Forms* are automatically added to it below any other *Menu* or *MenuItem* objects that the application has created directly. This list is separated from the preceding items by a separator. The entry for the currently active *SubForm* is checked and the user may switch between *SubForms* by selecting from this list.

Note that the additional separator and the items representing the list of child forms are not Dyalog APL/W objects and may not be accessed by the application. If you prefer to maintain your own "window list" you should not use this property.

3.186 MapCols Property

Applies To: *ImageList*

Description

The *MapCols* property specifies whether or not the button colours in bitmaps and icons in an *ImageList* are re-mapped to reflect the users colour preferences. If your bitmaps and icons represent buttons using the standard windows button colours, this property causes those colours to be changed to suit the user's own colour scheme.

MapCols is a single number with the value 0 (no colour mapping) or 1 (colours are automatically re-mapped). The default is 0.

If *MapCols* is 1, the following colour mappings are performed:

Colour	Description	Mapped to
0 0 0	Black	Button Text
128 128 128	Dark grey	Button Shadow
191 191 191	Light grey	Button Face
192 192 192	Light grey	Button Face
255 255 255	White	Button Highlight

3.187 Mask Property

Applies To: *Cursor*, *Icon*

Description

This property is used to specify how the bitmap for a *Cursor* or *Icon* interacts with the pixels of the screen when it is displayed.

When a *Cursor* or *Icon* is displayed, the colour of each pixel occupied by the object on the screen is determined by :

- The colour specified by *Bits* via *CMap*
- The value of *Mask*
- The existing colour of the screen pixel

Mask is a Boolean matrix with the same shape as the *Bits* property. See *Cursor* and *Icon* objects for further details.

3.188 MaskCol Property

Applies To: *Bitmap*, *Form*

Description

Specifies the transparent colour for a *Bitmap* or *Form*.

MaskCol may be an integer scalar or a 3-element integer vector.

If *MaskCol* is 0 (the default), no transparent colour is defined.

If *MaskCol* is a negative scalar, it specifies a standard Windows colour. See *BCol* for details.

Otherwise, *MaskCol* is a 3-element vector of integers in the range 0-255 that specifies the transparent colour in terms of RGB values (the intensity of the red, green and blue components of colour).

For a *Bitmap*, if *MaskCol* is non-zero, any pixels specified with the same colour will instead be displayed in whatever colour is underneath the *Bitmap*. This achieves similar behaviour to that of an *Icon*.

For a *Form*, if *MaskCol* is non-zero, any of the contents of the *Form* that are specified to be the same colour as *MaskCol* will be transparent. For example, if *MaskCol* is 255 0 0 (red), any red items contained in the *Form* will instead be transparent areas, displaying whatever is behind them on the screen. Mouse events generated over such transparent areas will be passed to any other windows behind them, and will not be reported on the *Form* itself.

3.189 Masked Property

Applies To: *ImageList*

Description

The *Masked* property specifies whether or not the *ImageList* will contain opaque or transparent images. It may be 0, 1 (the default), 2, or 3.

Masked must be established when the *ImageList* is created by `ImageList` and may not subsequently be altered. An inappropriate value of *Masked* will cause the images to be drawn incorrectly.

If *Masked* is 0, the *ImageList* expects opaque *Bitmap* objects.

If *Masked* is 1, the *ImageList* expects low-colour (4-bit or 8-bit) *Icon* objects whose transparency is defined by their *Mask* property.

If *Masked* is 2, the *ImageList* expects *Bitmap* or *Icon* objects whose alpha channel (the degree of transparency of each pixel) is encoded in its source file along with the colours.

If *Masked* is 3 and *Native Look and Feel*

is enabled, the behaviour is the same as if *Masked* were 2. If *Native Look and Feel* is not enabled, it behaves as if *Masked* were 1. This setting provides the greatest degree of portability for applications whose users may or may not have *Native Look and Feel* enabled. This value is used for the *ImageLists* on the *Dyalog Session CoolBars*.

3.190 MaxButton Property

Applies To: *Form*, *HTMLRenderer*, *SubForm*

Description

This property determines whether or not an object has a "maximise" button. Pressing this button will cause a *Form* or *HTMLRenderer* to be resized to occupy the entire screen, or a *SubForm* to occupy the entire area of its parent. Pressing it again will restore the object to its original size. *MaxButton* is a single number with the value 0 (no maximise button) or 1 (maximise button is provided). The default is 1.

Note that *MaxButton* is independent of *Sizeable*, that is, you can define an object that can be maximised but not resized. If any of the properties *MaxButton*, *MinButton*, *SysMenu* and *Sizeable* are set to 1, the object will have a title bar.

3.191 MaxDate Property

Applies To: *Calendar*, *DateTimePicker*

Description

The *MaxDate* property specifies the largest date that the user may select in a *Calendar* object or in the calendar drop-down of a *DateTimePicker*.

MaxDate is an *IDN* value. Its default value is 11249470 which is the maximum date that the *Calendar* can display.

3.192 MaxLength Property

Applies To: *ButtonEdit*, *Edit*, *Spinner*

Description

This property specifies the maximum number of characters that the user may enter in a single-line *Edit* object (*Style* 'Single') or in the edit field associated with a *Spinner*. It does not apply to a multi-line *Edit* object (*Style* 'Multi'). *MaxLength* does not limit the length of the vector that you may assign to the *Text* property using *Dyalog APL Language: Wc* or *Dyalog APL Language: Ws*. However, if you overfill the field in this way, the user must delete excess characters before the object will accept further input.

3.193 MaxSelCount Property

Applies To: *Calendar*

Description

The MaxSelCount property specifies the maximum number of contiguous days that the user may select in a *Calendar* object.

MaxSelCount is an integer whose default value is 7.

MaxSelCount is ignored unless the *Style* property of the *Calendar* object is set to 'Multi'.

3.194 MetafileObj Property

Applies To: *Clipboard*

Description

This property is used to copy graphical data to and from the Windows clipboard using the Windows Metafile format.

When you set the MetafileObj property of a *Clipboard* object to the name of the *Metafile* object using *Dyalog APL Language: Ws* its contents are copied to the Windows clipboard in Windows Metafile format.

To import a picture that has been stored in the Windows clipboard in Metafile format you use *Dyalog APL Language: Wg*. This returns a nested array whose elements correspond to the graphical components of the picture. Each of the elements of the array may be used as the arguments of *Dyalog APL Language: Wc* to draw the corresponding component of the picture. For example, if the picture stored in C:\MSOFFICE\CLIPART\BIRD.WMF is copied to the Windows clipboard, it may be imported into Dyalog APL/W as follows :

```
BIRD ← 'CL' ⎕WG 'MetafileObj'
ρBIRD
```

Each of the items in BIRD is a 2-element vector. The first element is a "dummy" object name which you may use or ignore as you wish. The second element is an array that defines a graphical object and is suitable as the right argument of *Dyalog APL Language: Wc*. For example :

```

234BIRD
POLY 191 397 FSTYLE 0 FILLCOL 0 0 0 ...

190 402
187 406
182 409
176 410
172 409
168 406
165 402
164 397
165 391
168 387
172 384
176 383
182 384
187 387
190 391
191 397
189 395
191 397

```

From this array, you can rebuild the imported picture component by component, either as a *Metafile* object or directly onto a *Form*, *Static* or another object. The following example draws the picture in a *Form* using the dummy names supplied.

```

'TEST' □WC 'FORM' ('Coord' 'User')
'TEST' □WS ('YRange' 0 1024)('XRange' 0 2048)
TEST.□WC/"BIRD

```

Notice that the co-ordinates of each of the graphical components are typically integers in a co-ordinate system that extends from 0 to 1024 in the y-direction and 0 to 2048 in the x-direction. The simplest way to draw the picture is therefore to set up the same co-ordinate system on a *Form* as in the example above.

3.195 MethodList Property

Applies To

Description

This property reports the names of all the methods supported by a particular COM object or instance of an OLE control. It is a vector of character vectors returned by □WG. It may not be set using □WC or □WS.

3.196 MinButton Property

Applies To: *Form*, *HTMLRenderer*, *SubForm*

Description

This property determines whether or not an object has a "minimise" button. Pressing this button will cause the object to be iconified. Pressing it again will restore the object to its original size. *MinButton* is a single number with the value 0 (no minimise button) or 1 (minimise button is provided). The default is 1.

Note that *MinButton* is independent of *Sizeable*, that is, you can define an object that can be minimised but not resized.

If any of the properties *MinButton*, *MaxButton*, *SysMenu*, and *Moveable* are set to 1, the object will have a title bar.

3.197 MinDate Property

Applies To: *Calendar*, *DateTimePicker*

Description

The *MinDate* property specifies the smallest date that the user may select in a *Calendar* or *DateTimePicker* object.

MinDate is an *IDN* value. Its default value is -109206 which is the minimum date that the *Calendar* can display.

3.198 MonthDelta Property

Applies To: *Calendar*, *DateTimePicker*

Description

The *MonthDelta* property specifies the number of months by which a *Calendar* object scrolls when the user clicks its scroll buttons.

MonthDelta is an integer or an empty vector (zilde). The latter means that the *Calendar* object scrolls by the number of months that are currently displayed in its window. This is the default.

3.199 Moveable Property

Applies To: *Form, HTMLRenderer, SubForm*

Description

This property determines whether or not an object can be moved by the user. It is a single number with the value 0 (object cannot be moved) or 1 (object is moveable). If any of the properties *MinButton*, *MaxButton*, *SysMenu*, and *Moveable* are set to 1, the object will have a title bar. However, an object with a title bar is not necessarily moveable.

3.200 MultiColumn Property

Applies To: *List*

Description

MultiColumn is Boolean and specifies whether or not a List object displays its items in a single column (0, the default) or in multiple columns (1). MultiColumn may only be set by `WC` and cannot be changed using `WS` after the object has been created. Note that a MultiColumn List will use the minimum number of columns that are required to make the items fit within it and will reconfigure itself automatically when resized.

The following example illustrates its use.

```
'F'WC'Form' 'MultiColumn List'('Size' 23 32)
'F.L'WC'LIST' AIRPORTS (0 0)(100 100)('MultiColumn' 1)
```



3.201 MultiLine Property

Applies To: *TabControl*, *ToolControl*

Description

The *MultiLine* property determines whether or not the tabs or buttons will be arranged in multiple flights or multiple rows/columns in a *TabControl* or *ToolControl* object.

MultiLine is a single number with the value 0 (single flight of tabs, or single row/column of buttons) or 1 (multiple flights of tabs or multiple rows/columns of buttons); the default is 0.

If *MultiLine* is 0 and there are more tabs or buttons than will fit in the space provided, the *TabControl* displays an *UpDown* which allows the user to scroll.

However, If *MultiLine* is 0 in a *ToolControl*, the buttons are clipped, and the user may have to resize the object to see them all.

See also: *Justify*, *TabSize*.

3.202 MultiSelect Property

Applies To: *TabControl*

Description

The *TabControl* property specifies whether or not the user can select more than one button in a *TabControl* at the same time, by holding down the Ctrl key when clicking.

MultiSelect is a single number with the value 0 (only 1 button may be selected) or 1 (more than one button may be selected); the default is 0.

MultiSelect applies only if the *Style* of the *TabControl* is 'Buttons' or 'FlatButton', and is ignored if *Style* is 'Tabs'.

Note that the *State* property of the associated *TabButton* object reports whether or not the button is selected.

3.203 NewLine Property

Applies To: *CoolBand*

Description

The `NewLine` property specifies whether or not a *CoolBand* occupies the same row as an existing *CoolBand*, or is displayed on a new line within its *CoolBar* parent.

`NewLine` is a single number with the value 0 (same row) or 1 (new row); the default is 1.

The value of `NewLine` in the first *CoolBand* in a *CoolBar* is always 1, even if you specify it to be 0.

When the user drags a *CoolBand* to another row, the value of its `NewLine` property, and that of any other *CoolBand* affected by the move, will change.

You may move a *CoolBand* to the previous or next row by changing its `NewLine` property (using `WS`) from 1 to 0, or from 0 to 1 respectively.

3.204 Note Property

Applies To: *Button*

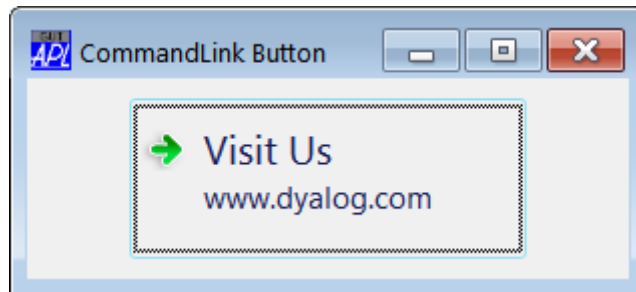
Description

The `Note` property applies only to a *Button* whose `Style` is `'CommandLink'`.

It is a character vector (by default empty) that specifies text to be displayed below the *Caption*.

EXAMPLE

```
'F' WC 'Form' 'CommandLink Button'
'F.clb' WC 'Button' 'Visit Us' ('Style' 'CommandLink')
F.clb.Size←80 200
F.clb.Note←'www.dyalog.com'
```



3.205 OLEControls Property

Applies To: *Root*

Description

The OLEControls property reports a list of the OLE Controls installed on your computer. This information is obtained from the Windows registry. Its value is a nested vector with one element per OLE Control. Each element is a vector of 2-element character vectors. The first is the name of the OLE Control; the second is its *class identifier*.

3.206 OLEServers Property

Applies To: *Root*

Description

The OLEServers property is a read-only property that reports the names and CLSIDs of all the OLE Automation servers installed on your computer. This information comes from the Windows registry.

Its value is a nested vector with one element per OLE Server.

Each element is a vector of 2-element character vectors. The first is the name of the OLE Server; the second is its class identifier or CLSID which is a type of *GUID*.

3.207 OnTop Property

Applies To

Description

This property may be used to cause a *Form* or *SubForm* to be displayed on top of all other windows, even those owned by other applications.

Normally, *Forms* are brought to the front when they receive the input focus. *Forms* that do not have the input focus may be partially obscured by the one that does. If OnTop is set to 1, the *Form* or *SubForm* remains at the front even if it doesn't have the input focus. Indeed, it may partially obscure the *Form* that does have the focus. The default value is 0 (normal).

More than one *Form* may have *OnTop* set to 1. If so, these *Forms* appear on top of all others, but may overlap one another. Other applications may also have windows with this property.

For a graphical object, the *OnTop* property controls how it is drawn in a *Grid* relative to the grid lines and cell text. *OnTop* is applicable only if the graphic is the child of a *Grid* and is otherwise ignored.

0	Graphical object is drawn behind grid lines and cell text
1	Graphical object is drawn on top of grid lines but behind cell text
2	Graphical object is drawn on top of grid lines and cell text

3.208 Orientation Property

Applies To: *Printer*

Description

The *Orientation* property specifies the orientation of the paper on a *Printer* object. It is a simple character vector which is either 'Portrait' or 'Landscape'. When you create a *Printer* object, the default value of the *Orientation* property is determined by the current setting for the corresponding printer device.

The effect of changing *Orientation* using `□ws` is to spool the current page (effectively the same as sending a *NewPage* event) and then to change the orientation of the paper. Note that the values of the first 2 elements of the *DevCaps* property change accordingly. You may also set *Orientation* when you create the *Printer* object with `□wc`. In neither case does the global setting for the printer device change.

3.209 OtherButton Property

Applies To: *ColorButton*

Description

The *OtherButton* property is Boolean and specifies whether or not the user can select a colour from a *ColorButton* object using the Windows colour selection dialog box.

If *OtherButton* is 1 (the default), the final row of the colour selection drop-down contains a button labelled "Other...". If the user clicks this button, the standard

Windows colour selection dialog box is displayed, allowing the user to select any colour that the computer can render.

If OtherButton is 0, the button labelled "Other..." is not present and the user is restricted to the choice of colours provided by the *DefaultColors* property.

3.210 OverflowChar Property

Applies To: *Grid*

Description

The OverflowChar property specifies the character to be displayed in place of the digits when a numeric value cannot be displayed in its entirety in a Grid cell.

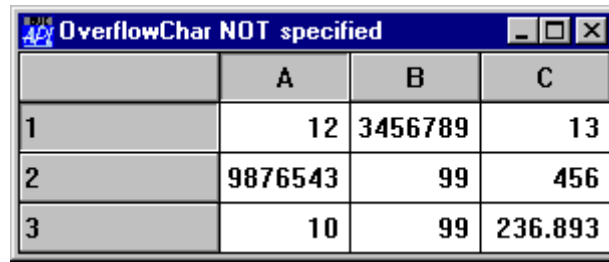
If the value of OverflowChar is an empty vector (the default) the data in a numeric cell is simply clipped if it is too wide to fit in the cell.

FOR EXAMPLE

```
'F'[]WC'Form'('Coord' 'Pixel')('Size' 101 296)
'F'[]WS'Caption' 'OverflowChar Property'
DATA←3 3p12 123456789 13 9876543 99 456 10 99 1236.893
'F.G'[]WC'Grid'DATA(0 0)(101 296)
'F.G'[]WS'CellWidths' 65
'F.G'[]WS'OverflowChar' '#'
```

	A	B	C
1	12	#####	13
2	#####	99	456
3	10	99	####.###

The same Grid without OverflowChar being defined appears as follows. Notice how the numbers have been truncated



	A	B	C
1	12	3456789	13
2	9876543	99	456
3	10	99	236.893

3.211 PName Property

Applies To: *Font*, *Printer*

Description

This property is a character vector that specifies the face name for a *Font* object, or the printing device associated with a *Printer*. It is case-independent.

For a *Printer*, PName contains the description of the printer followed by a comma (,) and then the device to which it is attached.

EXAMPLE

```
'PR1' WC 'Printer'
'PR1' WC 'PName'
HP Universal Printing PS,hp4200
```

3.212 PageActive Property

Applies To: *PropertySheet*

Description

The PageActive property specifies the name of the current *PropertyPage* in a *PropertySheet*. You may select a particular page by setting this property or by generating a *PageActivate* event.

See also or *PageActiveObject*.

3.213 PageActiveObject Property

Applies To: *PropertySheet*

Description

The `PageActiveObject` property specifies a ref to the current *PropertyPage* in a *PropertySheet*. You may select a particular page by setting this property or by generating a *PageActivate* event.

See also or *PageActive*.

3.214 PageSize Property

Applies To: *Form*, *Scroll*, *SubForm*

Description

For a *Form* and *SubForm*, the `PageSize` property is a 2-element integer vector which specifies the size of the thumb in the vertical and horizontal scrollbars respectively.

For a *Scroll* object it is a single integer.

If `PageSize` is 0 (the default) it specifies the default thumb. Otherwise, `PageSize` is expressed in proportion to the corresponding value of *Range*. For example, if *Range* is 1000, setting `PageSize` to 100 will obtain a thumb which is approximately 10% of the height or length of the scrollbar.

3.215 PageWidth Property

Applies To: *RichEdit*

Description

The `PageWidth` property specifies the width of the page in a *RichEdit* object and is the dimension that is used to apply text wrapping and paragraph formatting to the text in the object. `PageWidth` is a single integer value specified in *Twips*.

The default value of `PageWidth` is 0 which disables text wrapping. You may find it convenient to set `PageWidth` to the width of the *RichEdit* window or to a value that is appropriate for your printer.

3.216 PaperSize Property

Applies To: *Printer*

Description

The `PaperSize` property specifies the size of paper to be used for printing

`PaperSize` may be a character vector containing the name of the paper size (for example, 'Legal 8 1/2 x 14 in' or 'A4 210 x 297 mm') or a 2-element integer vector that specifies the desired height and width of the paper in tenths of a millimetre (for example, 3556 2159 or 2970 2099).

The default value of `PaperSize` is the name of the paper size associated with the current printer settings.

You can obtain a list of supported paper sizes from the *PaperSizes* property.

3.217 PaperSizes Property

Applies To: *Printer*

Description

The `PaperSizes` property is a read-only property that provides the names and dimensions of the various different paper sizes supported by the printer associated with the `Printer` object.

`PaperSizes` is a nested vector of 2-element vectors which contain the name, and height and width of each paper size respectively. Dimensions are reported in tenths of a millimetre.

You may set or query the current paper size using the *PaperSize* property.

3.218 PaperSource Property

Applies To: *Printer*

Description

The `PaperSource` property is a character vector that specifies the name of the paper bin to be used as the paper source for printing.

An empty character vector (the default) means the default bin, Otherwise, `PaperSource` must be a member of the *PaperSources* property.

3.219 PaperSources Property

Applies To: *Printer*

Description

The PaperSources property is a read-only property that provides the names of the paper bins installed on the printer associated with the Printer object. It is a vector of character vectors.

You may select which of the bins is to be used by specifying the *PaperSource* property.

3.220 ParaFormat Property

Applies To: *RichEdit*

Description

The ParaFormat property describes the current paragraph format or the paragraph format of the currently selected text in a *RichEdit* object. It is a 6-element nested array structured as follows:

[1]	A character vector that specifies the text alignment. This may be 'Left' (the default), 'Right' or 'Centre'
[2]	The size of the indentation of the first line in the paragraph measured from the left margin in <i>Twips</i> .
[3]	The size of the horizontal offset of the start of the second and subsequent lines. This is measured in <i>Twips</i> relative to the first line indentation specified in element [2].
[4]	The size of the right indentation measured in <i>Twips</i> from the right margin.
[5]	An integer value specifying the bullet/numbering option. 0 mean no numbering, 1 means bullets.
[6]	An integer vector specifying the size of any tab stops measured in <i>Twips</i> from the left margin and specified in ascending order.

If there is no text selected, ParaFormat specifies the *current* paragraph formatting format, that is, that which will be used to format the current (and subsequent) lines of characters that the user enters. If there is text selected ParaFormat specifies the

paragraph formatting of the selected block of text. If the format is not strictly homogeneous, ¶WG will report the format of the first paragraph in the selected block

(¶WS 'ParaFormat' ...) will set the format of the currently selected block of text. To set the format of an arbitrary block of text you must select it first using (¶WS 'SelText' ...).

3.221 Password Property

Applies To: *ButtonEdit, Edit, Spinner*

Description

This property specifies the character that is echoed when a user enters data into a single-line *Edit* object (*Style 'Single'*). It does not apply to a multi-line object (*Style 'Multi'*). If Password is empty (the default) the character echoed is the same as the character the user entered. If Password is set to (say) the asterisk character (*), the object will display asterisks as the user types into it.

3.222 PathWordBreak Property

Applies To: *ComboEx*

Description

If set, the edit control portion of the ComboEx will use the forward slash (/), back slash (\), and period (.) characters as word delimiters. This makes keyboard shortcuts for word-by-word cursor movement (Ctrl + arrow keys) effective in path names and URLs.

3.223 Picture Property

Applies To

Description

The Picture property specifies a bitmap, icon, or other image for an object.

For *Button, Form, Group, MDIClient, Static, StatusBar, StatusField, SubForm, SM, TabBar* or *ToolBar*, this property specifies the name of, or ref to, a *Bitmap, Icon, or Metafile* which is drawn as a *background* on the object. Other controls and graphical objects are drawn on top of this background.

When it refers to a *Metafile*, the *Picture* property specifies the name of, or ref to, the *Metafile* to be drawn in the object. When it refers to a *Bitmap* or *Icon*, the value of the *Picture* property is a 2-element vector whose elements specify the name of, or ref to, the *Bitmap*, or *Icon*, and the manner in which it is displayed. This is specified as an integer as follows:

0	The <i>Bitmap</i> or <i>Icon</i> is drawn in the top left corner of the object.
1	The <i>Bitmap</i> or <i>Icon</i> is tiled (replicated) to fill the object.
2	The <i>Bitmap</i> is scaled (up or down) to fit exactly in the object. This setting does not apply to an <i>Icon</i> whose size is fixed.
3	The <i>Bitmap</i> or <i>Icon</i> is drawn in the centre of the object. This is the default. Note that the centre of the <i>Bitmap</i> is positioned over the centre of the object, so that you see the middle portion of a <i>Bitmap</i> that is larger than the object in which it is displayed.

For example, the following statements produce a *Form* filled with the CARS bitmap.

```
'CARS' □WC 'Bitmap' 'C:\WINDOWS\CARS'
'f1' □WC 'Form' ('Picture' 'CARS' 1)
```

An easy way to provide a customised pushbutton is to create a *Button* whose *Picture* property specifies the name of, or ref to, a *Bitmap* or *Icon*, using drawmode 3 (the default). This causes the corresponding bitmap or icon to be drawn in the centre of the *Button*. So long as the *Button* is larger than the bitmap or icon, its borders (which give it its 3-dimensional appearance and "pushbutton" behaviour) will be unaffected.

Note that if *Picture* is set on a *Button* whose *Style* is 'Radio' or 'Check', the *Button* assumes pushbutton appearance, although its radio/check behaviour is preserved.

For an *Image* object, the *Picture* property specifies the name of, or ref to, a *Bitmap*, *Icon* or *Metafile* object to be drawn, or a vector of names or refs. The *Image* is a graphical object and is drawn *on top of* the background. It does not support the drawmode options provided by the objects in which *Picture* specifies the background.

For the *Clipboard* object, *Picture* is a "set-only" property that allows you to place a specified *Bitmap* object into the Windows clipboard. To place a *Metafile* object into the clipboard, use its *Metafile* property.

3.224 Points Property

Applies To

Description

This property specifies the co-ordinates for a graphics object. It may define a single set of co-ordinates, or be a nested scalar or a nested vector each element of which contains a set of co-ordinates.

Each set of co-ordinates may be:

- a 2-column numeric matrix containing y-values in column 1 and x-values in column 2. **or**
- a 2-element numeric vector whose first element specifies y-values and whose second element specifies x-values. The two elements must be of equal length unless one or both is a scalar in which case scalar extension applies.

For further details, see the specifications for the relevant objects.

3.225 Popup Property

Applies To: *SysTrayItem*, *ToolButton*

Description

The Popup property specifies the name of, or ref to, a (popup) *Menu* object that is associated with a *SysTrayItem* or *ToolButton*.

Note that Popup is ignored unless *Style* is set to 'DropDown'.

3.226 Posn Property

Applies To

Description

With the exception of *Menu*, *MenuItem* and *Separator* objects, Posn is a 2-element numeric vector specifying the y-position and x-position respectively of the top-left corner of the object relative to its parent. For a *Form*, Posn specifies its position on the screen. The units are defined by the *Coord* property.

When specifying Posn for *Dyalog APL Language: Wc*, you can allow the y-position or x-position to assume a default value by giving the corresponding element a value of $\emptyset$.

Using *Dyalog APL Language: Ws*, if you want to set the y-position, but not the x-position, or vice-versa, you should specify $\emptyset$ for the item you don't want to change.

For *Menu*, *MenuItem* and *Separator* objects, *Posn* is a single integer that specifies the position at which the object is to be **inserted** in its parent. For example, to add a new *MenuItem* between the third and fourth items in an existing *Menu*, you would specify its *Posn* as 4. For these objects, the value of *Posn* returned by *Dyalog APL Language*: *Wg* is the current index of the object within its parent.

3.227 PrintList Property

Applies To: *Root*

Description

This property provides a list of the printers that are installed on your computer system, that is, those listed when you select "printers" from the MS-Windows Control Panel. It is a "read-only" property of the *Root* object ' '. ' '.

PrintList is a vector of character vectors. Each item in *PrintList* contains the name of an installed printer followed by a comma (,) and then the name of the device to which it is attached. The first item in *PrintList* is the default system printer.

EXAMPLE

```

ρ ' '⊂WG'PrintList'
6
]display ⚡ ' '⊂WG'PrintList'
→
|KODAK ESP-3 AiO,USB001|
↑ ' '⊂WG'PrintList'
KODAK ESP-3 AiO,USB001
Send To OneNote 2007,Send To Microsoft OneNote Port:
Microsoft XPS Document Writer,XPSPort:
Microsoft Office Document Image Writer,Microsoft Document Imaging
Writer Port:
Fax,SHRFAX:
Auto Canon MP600 Printer on DIMENSION5150,\
\DIMENSION5150\Canon

```

3.228 PrintRange Property

Applies To: *Printer*

Description

The `PrintRange` property specifies the range of pages to be printed.

`PrintRange` may be an empty character vector (the default), or `'All'`, either of which will cause all pages to be printed.

Alternatively, `PrintRange` may be a 3 or 4-element nested array whose items are:

[1]	'Pages'
[2]	Start page (integer)
[3]	End page (integer)
[4]	Maximum number of pages (integer)

In this case, printing starts at the page specified to be the Start page, and ends at the page specified by End page or after the Maximum number of pages has been reached, whichever is sooner.

3.229 ProgressStyle Property

Applies To: *ProgressBar*

Description

The `ProgressStyle` property specifies the appearance of a *ProgressBar* control.

`ProgressStyle` is a character vector that may be `'Normal'`, `'Smooth'` or `'Marquee'`. Its value is effective only when the object is created with `□wc`. Changing `ProgressStyle` with `□ws` has no effect on the appearance or behaviour of the *ProgressBar*.

If `ProgressStyle` is `'Normal'`, the highlight in the centre of the *ProgressBar* is displayed as a broken bar. This is the default.

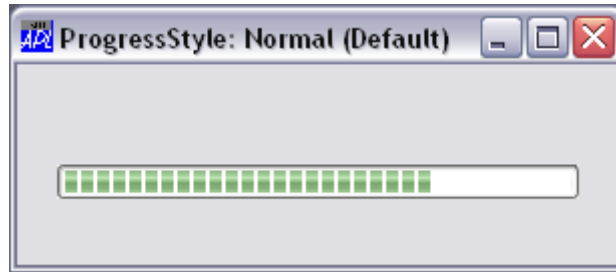
If `ProgressStyle` is `'Smooth'`, the highlight in the centre of the *ProgressBar* is displayed as a solid block of colour. This style only applies if *Windows Classic Theme* is in use. If not, it will be as if `'Normal'` were specified.

If `ProgressStyle` is `'Marquee'`, the highlight in the centre of the *ProgressBar* is displayed as a broken bar that moves continuously from left to right. The speed is controlled by the *Interval* Property which determines the frequency in milliseconds with which the highlight is redrawn, each time further along the *ProgressBar*. The special value of `-1` causes the animation to stop.

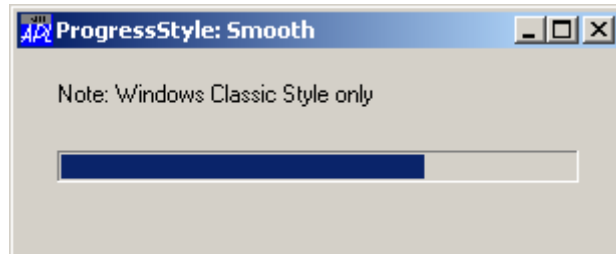
****Note that this feature only applies if Native Look and Feel**

is enabled.** If not, 'Marquee' will produce the same behaviour as 'Normal'.

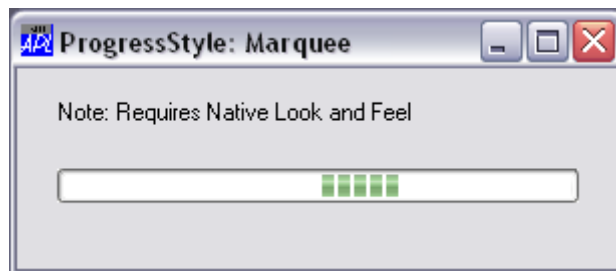
The pictures below illustrate the appearance of the different values of ProgressStyle.



ProgressStyle Normal (the default)



ProgressStyle Smooth (Windows Classic Theme only)



ProgressStyle Marquee (requires Native Look and Feel)

3.230 PropList Property

Applies To

Description

This is a "read-only" property that supplies a list of all other properties which are applicable to the object in question. The list is returned as a vector of character vectors in the order in which the corresponding properties are expected by *Dyalog APL Language: Wc* and *Dyalog APL Language: Ws*.

EXAMPLE

```
'F'      □WC 'Form'
'F.MB'   □WC 'MenuBar'
'F.MB'   □WG 'PropList'
Type Visible FontObj Data EdgeStyle MDIMenu PropList
```

3.231 QueueEvents Property

Applies To: *OCXClass*, *OLEClient*

Description

The QueueEvents property specifies whether or not incoming events generated by an COM object are queued. It is a Boolean value where the (default) value 1 specifies that events are queued, and 0 that they are not.

If QueueEvents is 1, the result (if any) of your callback function is not passed back to the COM object but is discarded. Thus you cannot, for example, inhibit or modify the default processing of the event by the COM object.

If QueueEvents is 0, the following applies.

- The callback function attached to the event is executed immediately, even if there are other APL events before it in the internal event queue. This immediate execution means that your callback can fire during the execution of any other function, including a callback function on an APL event. You must therefore take care that the callback makes no references to objects that may be shadowed.
- The result of your callback function is then passed back to the COM object. In this situation, it is essential that the callback is not interrupted by other events from the same, or another instance, of an COM object.
- To prevent APL itself from yielding to Windows, the Yield property is temporarily set to 0 while the callback is run. For the same reason, the tracing of a callback function, that is run immediately in this way, is disabled.

However, you must yourself also ensure that your own code does not yield. This means that you may not perform any operation in your callback that would yield to Windows; these include:

- `DL`
- certain uses of `NA`
- external function calls to Auxiliary Processors

If your callback does yield to Windows, thereby allowing another COM object event to arrive, this second event and any subsequent events that arrive during the execution of the callback are queued and will be processed later. These events may therefore not be modified by their callback functions.

3.232 RTFText Property

Applies To: *Clipboard*, *RichEdit*

Description

The RTFText property is used to set or retrieve the contents of a *Clipboard* or a *RichEdit* object in rich text format (RTF). It is always a character vector.

3.233 Radius Property

Applies To: *Circle*, *Rect*

Description

For a *Circle* object, this property is a single number that specifies the radius of the circle/arc or a numeric vector that specifies the radii of a set of circles/arcs.

For a *Rect* object, Radius is a 2-element vector that specifies the curvature of the corners of the rectangle or set of rectangles to be drawn. The curvature is defined in terms of the vertical and horizontal radii of an ellipse. The first element of Radius defines the radius vertically, the second horizontally. If more than one rectangle is involved, either or both of the elements of Radius may be vectors. The default value is (0,0) which gives square corners.

3.234 RadiusMode Property

Applies To: *Circle, Root*

Description

A perfectly round circle can only be drawn if the diameter is an odd number of pixels. The RadiusMode property specifies whether or not a circle is adjusted by a single pixel, if necessary, so as to appear perfectly round.

If RadiusMode is 1 or -1, and the diameter is an even number of pixels, the circle is actually drawn with a diameter of 1 pixel more or less than specified. If RadiusMode is 0 (the default), no such adjustment is made.

RadiusMode may be set on the Root object to be inherited by all Circle objects.

3.235 Range Property

Applies To: *Form, Scroll, SubForm*

Description

This property determines the maximum value of the thumb in a scrollbar (the minimum value is always 1). This may be any positive integer value that is greater than 1.

For a *Scroll* object Range is a single number. For a *Form* or *SubForm* object, Range is a 2-element vector which specifies the maxima for the *Form*'s vertical and horizontal scrollbars respectively.

3.236 ReadOnly Property

Applies To: *Button, ButtonEdit, Edit, Spinner*

Description

This property specifies whether or not the user may alter the text in an object. The default value of ReadOnly is 0 which allows the user to alter text.

If you set ReadOnly to 1, a cursor is displayed in the object, the user may navigate around the text in the usual manner with the mouse and/or the keyboard and select

text and copy it to the clipboard. However, all input that would otherwise change the data is ignored.

For a Button object with *Style* 'Radio' or 'Check', setting *ReadOnly* to 1 prevents the user from changing the state of the *Button*, although mouse and other events will still be reported.

3.237 RealSize Property

Applies To: *Metafile*

Description

There are several distinct types of Windows metafiles. A *placeable* metafile is one that carries with it its *suggested size*. Certain programs (such as Word for Windows) only support placeable metafiles.

The *RealSize* property specifies the suggested size of a Metafile in units of 0.01mm. Thus to make a placeable Metafile with a suggested size of 20 x 10 cm, you would set *RealSize* to (20000 10000).

The *RealSize* property is not used or required by Dyalog and is provided only to enable you to make and save a new metafile that is placeable. If you create a Metafile object from a file, the value of *RealSize* will be obtained from the value recorded in the file (if it is placeable). Otherwise, *RealSize* will be (0 0). If so, you must set *RealSize* to make it placeable. Each element of *RealSize* must be an integer in the range 0-144745.

3.238 Redraw Property

Applies To

Description

The *Redraw* property specifies whether or not APL automatically redraws an object when it is exposed or when any of its properties change in a way that would affect its appearance.

The value reported by the *Redraw* property is a Boolean value; 1 means that APL automatically redraws the object when necessary (the default); 0 means that APL does not redraw the object.

Setting *Redraw* to 0 or 1 affects only whether or not APL will redraw the object from then on.

Setting `Redraw` to 2 has the same effect as setting it to 1, but the object is also redrawn immediately. Note that no child object is redrawn.

Setting `Redraw` to 3 has the same effect as setting it to 1, but the object and its children are redrawn immediately.

3.239 RemoteAddr Property

Applies To: *TCPSocket*

Description

The `RemoteAddr` property is a character vector that specifies the IP address of the remote computer.

`RemoteAddr` may only be specified by a client *TCPSocket* that is intended to make a connection with a server. Furthermore, it must be specified in the `□WC` statement that creates the *TCPSocket* object and it may not subsequently be changed using `□WS`.

You may use either `RemoteAddr` or `RemoteAddrName` to identify the remote computer. If you know its IP address, it is normally quicker to specify `RemoteAddr`. If you specify both properties, the value of `RemoteAddrName` will be ignored.

For a server *TCPSocket*, `RemoteAddr` is determined by the IP address of the connecting process and is a read-only property.

3.240 RemoteAddrName Property

Applies To: *TCPSocket*

Description

The `RemoteAddrName` property is a character vector that specifies the host name of the remote computer to which you wish to make a connection.

`RemoteAddrName` may only be specified by a client *TCPSocket* that is intended to make a connection with a server. Furthermore, it must be specified in the `□WC` statement that creates the *TCPSocket* object and it may not subsequently be changed using `□WS`.

When the specified host name has been resolved to an IP address, the *TCPSocket* will generate a `TCPGotAddr` event and update the value of `RemoteAddr` accordingly.

Note that you may use *either RemoteAddr or RemoteAddrName* to identify the remote computer. If you know its IP address, it is normally quicker to specify *RemoteAddr*. If you specify both properties, the value of *RemoteAddrName* will be ignored.

For a server *TCP Socket*, you may not specify *RemoteAddrName* and `WG` returns an empty character vector.

3.241 RemotePort Property

Applies To: *TCP Socket*

Description

The *RemotePort* property is a scalar integer in the range 1-65536 that identifies the *port number* associated with a service on a remote computer.

RemotePort may only be specified by a client *TCP Socket* that is intended to make a connection with a server. Furthermore, it must be specified in the `WG` statement that creates the *TCP Socket* object and it may not subsequently be changed using `WS`.

Note that you may use *either RemotePort or RemotePortName* to identify the remote service. If you know the port number, it is normally quicker to specify *RemotePort*. However unless it is a *well known port number*, the use of a port name is generally more flexible. If you specify both properties, the value of *RemotePortName* will be ignored.

For a server *TCP Socket*, *RemotePort* is determined by the *port number* of the connecting process and is a read-only property.

3.242 RemotePortName Property

Applies To: *TCP Socket*

Description

The *RemotePortName* property is a character vector that specifies the port name of the remote service to which you wish to make a connection.

RemotePortName may only be specified by a client *TCP Socket* that is intended to make a connection with a server. Furthermore, it must be specified in the `WG` statement that creates the *TCP Socket* object and it may not subsequently be changed using `WS`.

When the specified port name has been resolved to a port number, the *TCPSocket* will generate a *TCPGotPort* event and update the value of *RemotePort* accordingly.

Note that you may use *either RemotePort or RemotePortName* to identify the remote service. If you know the port number, it is normally quicker to specify *RemotePort*. However unless it is a *well known port number*, the use of a port name is generally more flexible. If you specify both properties, the value of *RemotePortName* will be ignored.

For a server *TCPSocket*, you may not specify *RemotePortName* and `WG` returns an empty character vector.

3.243 ReportBCol Property

Applies To: *ListView*

Description

In Report View, the *ReportBCol* property is either a scalar or a matrix that specifies the background colours for each item displayed in a *ListView* object .

Its first column refers to the *Items* themselves, and subsequent columns to the elements of *ReportInfo*.

That is, if non-scalar, $(pReportBCol) \leftrightarrow (0 \ 1 + pReportInfo)$

Each element of *ReportBCol* is either an integer colour value or a 3-element of RGB colour indices.

For further information, see "*BCol*".

3.244 ReportImageIndex Property

Applies To: *ListView*

Description

The *ReportImageIndex* property is an integer scalar or matrix that specifies the images to be displayed alongside each item in a *ListView* object in Report View.

If it is a matrix, its first column specifies the indices of the icons to be displayed against the *Items* of the *ListView*, overriding the icons specified by *ImageIndex*, and its

subsequent columns specify the indices of the icons to be displayed against the elements of *ReportInfo*.

That is, if non-scalar, $(pReportImageIndex) \leftrightarrow (0 \ 1 + pReportInfo)$

Each element of *ReportImageIndex* specifies an index into the *ImageList* object specified by the *ImageListObj* property.

3.245 ReportInfo Property

Applies To: *ListView*

Description

The *ReportInfo* property is a matrix that is displayed alongside each item in a *ListView* object when its *View* property is 'Report'. Each element of the matrix may be a character vector or a number.

The information is displayed in a grid format, the first column of which contains the item labels and their icons. Subsequent columns of the grid are defined by the corresponding columns of *ReportInfo*. The alignment of the columns is specified by the *ColTitleAlign* property.

3.246 ResizeColTitles Property

Applies To: *Grid*

Description

This property determines whether or not the user may alter the height of the column titles in the *Grid*. It is either 1, which indicates that the height of the column titles is adjustable by the user, or 0 which means that it is not.

If the height of the column titles is adjustable, the cursor changes to a double headed arrow when the mouse pointer is placed over the top border of the first row title. The user may resize the column titles by dragging this border. The user may also resize the column titles by double-clicking over this border. This causes the column titles to be resized to fit the data and the height of the column titles is automatically adjusted to display the tallest heading in any of its columns. Either operation generates a *SetRowSize* event. The value of the row number reported by the event is -1.

Note that the user may cause the column titles to disappear altogether by dragging them to a zero height. Once this has been done, the row titles cannot be restored.

3.247 ResizeCols Property

Applies To: *Grid*

Description

This property determines whether or not the user may resize columns in the *Grid*. It is a Boolean scalar or vector with one element per column. A value of 1 indicates that the corresponding column is resizable by the user. A value of 0 means that the corresponding column may not be resized by the user.

If a column is resizable, the cursor changes to a double headed arrow when the mouse pointer is placed over the right-hand border of the column title. The user may resize the column by dragging this border. The user may also resize a column by double-clicking over its right-hand border. This causes the column to be resized to fit the data and the width of the column is automatically adjusted to display the widest value in any of its cells. Either operation generates a *SetColSize* event.

Note that the user may cause the column to disappear altogether by dragging it to a zero width. Once this has been done, this column may only be restored if the column to its left is itself not resizable.

3.248 ResizeRowTitles Property

Applies To: *Grid*

Description

This property determines whether or not the user may alter the width of the row titles in the *Grid*. It is either 1, which indicates that the width of the row titles is adjustable by the user, or 0 which means that it is not.

If the width of the row titles is adjustable, the cursor changes to a double headed arrow when the mouse pointer is placed over the left-hand border of the first column title. The user may resize the row titles by dragging this border. The user may also resize the row titles by double-clicking over this border. This causes the row titles to be resized to fit the data and the width of the row titles is automatically adjusted to display the longest string in any of its rows. Either operation generates a *SetColSize* event. The value of the column number reported by the event is -1.

Note that the user may cause the row titles to disappear altogether by dragging them to a zero width. Once this has been done, the row titles cannot be restored.

3.249 ResizeRows Property

Applies To: *Grid*

Description

This property determines whether or not the user may resize rows in the *Grid*. It is a Boolean scalar or vector with one element per column. A value of 1 indicates that the corresponding row is resizable by the user. A value of 0 means that the corresponding row may not be resized by the user.

If a row is resizable, the cursor changes to a double headed arrow when the mouse pointer is placed over the lower border of the row title. The user may change the height of the row by dragging this border up and down. The user may also resize a row by double-clicking over its bottom border. This causes the row to be resized to fit the data and the height of the row is automatically adjusted to display the tallest value in any of its cells. Either operation generates a *SetRowSize* event.

Note that the user may cause the row to disappear altogether by dragging it to a zero height. Once this has been done, this row may only be restored if the row above it is itself not resizable.

3.250 Resolution Property

Applies To: *Printer*

Description

The Resolution property determines the print resolution.

You may set Resolution to 'Draft', 'Low', 'Medium' or 'High'.

Alternatively, you can set Resolution to a 2-element integer vector that specifies the desired number of dots per inch in the x (horizontal) and y (vertical) direction respectively.

The initial value reported by Resolution may be reported in either form (character vector or 2-element numeric vector) according to the current printer settings.

3.251 Resolutions Property

Applies To: *Printer*

Description

The Resolutions property is a read-only property that reports the available printer resolutions.

Resolutions is a vector of 2-element integer vectors each of which specifies the number of dots per inch in the x (horizontal) and y (vertical) directions respectively.

3.252 Rotate Property

Applies To: *Font*

Description

This property specifies the angle of rotation of the font measured in radians ($0 \rightarrow \pi/2$) from the x-axis in a counter-clockwise direction. Note that only TrueType fonts can be rotated. Rotated fonts are supported **only** for use with the *Text* object.

3.253 RowHiddenDepth Property

Applies To: *Grid*

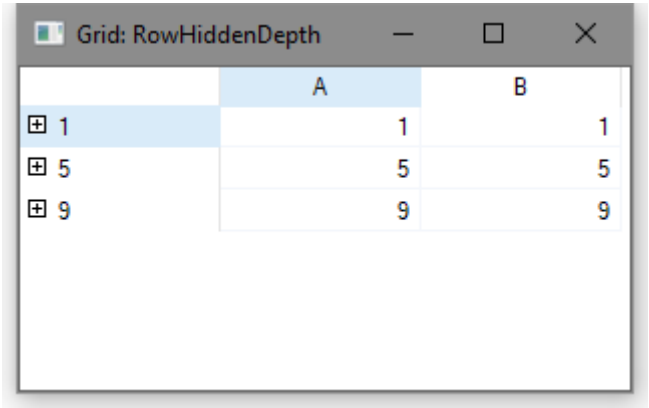
Description

The RowHiddenDepth property is read-only and identifies which rows of a Grid are currently hidden.

RowHiddenDepth is an integer vector with the same number of elements as there are rows in the Grid. The values in RowHiddenDepth indicate the current depth of the row in the visible hierarchy; that is, number of nodes that must be opened to display it. The value 0 means that the corresponding row is visible. The value 1 means that 1 node must be opened to display it; 2 means 2 nodes, and so forth.

EXAMPLE

```
'F'□WC'Form' 'Grid: RowHiddenDepth'
F.(Coord Size)←'Pixel'(163 306)
'F.G'□WC'Grid'(10 2p2/110)('Size'F.Size)
□←F.G.RowTreeDepth←10p0 1 2 3
0 1 2 3 0 1 2 3 0 1
```

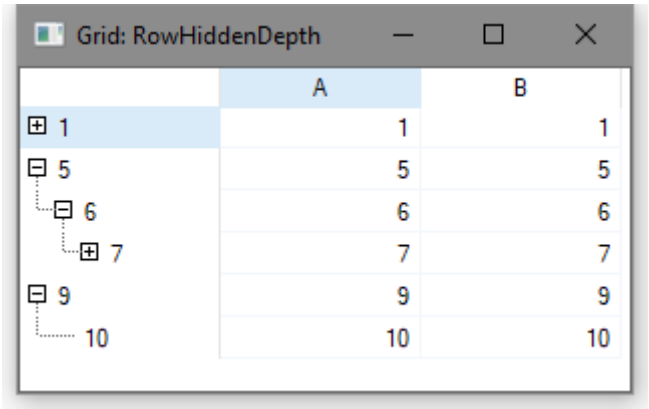


	A	B
1	1	1
5	5	5
9	9	9

With all nodes closed, RowHiddenDepth is the same as RowTreeDepth.

```
F.G.RowHiddenDepth
0 1 2 3 0 1 2 3 0 1
```

The next picture shows the Grid after the user has opened nodes 5,6 and 9.



	A	B
1	1	1
5	5	5
6	6	6
7	7	7
9	9	9
10	10	10

```
F.G.RowHiddenDepth
0 1 2 3 0 0 0 1 0 0
```

3.254 RowLineTypes Property

Applies To: *Grid*

Description

This property specifies the appearance of the horizontal grid lines in a *Grid* object.

RowLineTypes is an integer vector, whose length is normally equal to the number of rows in the *Grid*. Each element in RowLineTypes specifies an index into the *GridLineFCol* and *GridLineWidth* properties, thus selecting the colour and width of the horizontal grid lines.

For example, if RowLineTypes[1] is 3, the first horizontal grid line in the *Grid* is displayed using the colour specified by the 3rd element of *GridLineFCol*, and the width specified by the 3rd element of *GridLineWidth*.

Note that RowLineTypes is not ± 10 dependent, and the value 0 is treated the same as the value 1; both selecting the *first* colour and line width specified by *GridLineFCol* and *GridLineWidth* respectively.

The default value of RowLineTypes is an empty numeric vector ($\emptyset$). If so, all horizontal grid lines are drawn using the first element of *GridLineFCol* and *GridLineWidth*.

A horizontal grid line is drawn along the bottom edge of its associated row. One pixel is drawn *inside* the row of cells; additional pixels (if any) are drawn *between* that row of cells and the next one below.

3.255 RowTitleAlign Property

Applies To: *Grid*

Description

The RowTitleAlign property specifies the alignment of row titles in a *Grid*. It is either a simple character vector, or a vector of character vectors with one element per row.

An element of RowTitleAlign may be: 'Top', 'Bottom', 'Left', 'Right', 'Centre', 'TopLeft', 'TopRight', 'BottomLeft', Or 'BottomRight'.

Note that both spellings 'Centre' and 'Center' are accepted.

3.256 RowTitleBCol Property

Applies To: *Grid*

Description

The RowTitleBCol property specifies the background colour of the row titles in a *Grid* object

RowTitleBCol may be a scalar that specifies a single background colour to be used for all of the row titles, or a vector that specifies the background colour of each of the row titles individually. An element of RowTitleBCol may be an enclosed 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or it may be a scalar that defines a standard Windows colour element (see *BCol* for details). Its default value is 0 which obtains the colour defined for Button Face.

3.257 RowTitleDepth Property

Applies To: *Grid*

Description

RowTitleDepth specifies the structure of a set of hierarchical row titles.

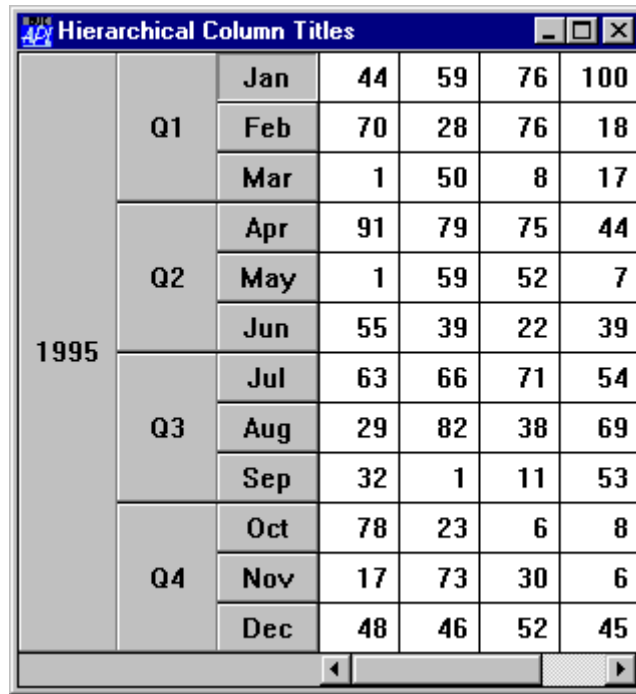
It is an integer vector with the same length as the RowTitles property. A value of 0 indicates that the corresponding element of RowTitles is a top-level title. A value of 1 indicates that the corresponding title is a sub-title of the most recent title whose RowTitleDepth is 0; a value of 2 indicates that the corresponding title is a sub-title of the most recent title whose RowTitleDepth is 1, and so forth. For example:

```
'F'□WC'Form'('Coord' 'Pixel')('Size' 318 310)
'F'□WS'Caption' 'Hierarchical Column Titles'
'F.G'□WC'Grid'(?12 4p100)(0 0)(318 310)
'F.G'□WS('TitleWidth' 150)('TitleHeight' 0)
'F.G'□WS'CellWidths' 40

Q1←'Q1' 'Jan' 'Feb' 'Mar'
Q2←'Q2' 'Apr' 'May' 'Jun'
Q3←'Q3' 'Jul' 'Aug' 'Sep'
Q4←'Q4' 'Oct' 'Nov' 'Dec'
RT←('1995'),Q1,Q2,Q3,Q4
RD←0,16p1 2 2 2

'F.G'□WS('RowTitles'RT)('RowTitleDepth'RD)

'F.G'□WS'RowTitleAlign' 'Centre'
```



Hierarchical Column Titles						
1995	Q1	Jan	44	59	76	100
		Feb	70	28	76	18
		Mar	1	50	8	17
	Q2	Apr	91	79	75	44
		May	1	59	52	7
		Jun	55	39	22	39
	Q3	Jul	63	66	71	54
		Aug	29	82	38	69
		Sep	32	1	11	53
	Q4	Oct	78	23	6	8
		Nov	17	73	30	6
		Dec	48	46	52	45

Note that the *LockRows* method is not supported in combination with hierarchical row titles.

3.258 RowTitleFCol Property

Applies To: *Grid*

Description

The RowTitleFCol property specifies the colour of the row titles in a *Grid* object

RowTitleFCol may be a scalar that specifies a single colour to be used for all of the row titles, or a vector that specifies the colour of each of the row titles individually. An element of RowTitleFCol may be an enclosed 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or it may be a scalar that defines a standard Windows colour element (see *BCol* for details). Its default value is 0 which obtains the colour defined for Button text.

3.259 RowTitles Property

Applies To: *Grid*

Description

This property specifies the headings that are displayed to the left of the rows in a *Grid* object. If specified, it must be a vector of character vectors or matrices whose length is the same as the number of rows implied by the *Values* property. The default value of *RowTitles* is an empty character vector. In this case, the system displays the row numbers.

To disable the display of row titles in a *Grid*, you should set the *TitleWidth* property to 0.

3.260 RowTreeDepth Property

Applies To: *Grid*

Description

The *RowTreeDepth* property specifies the structure of the rows in a *Grid* object. It is either a scalar 0 or an integer vector of the same length as the number of rows in the *Grid*. *RowTreeDepth* is similar to the *Depth* property of the *TreeView* object.

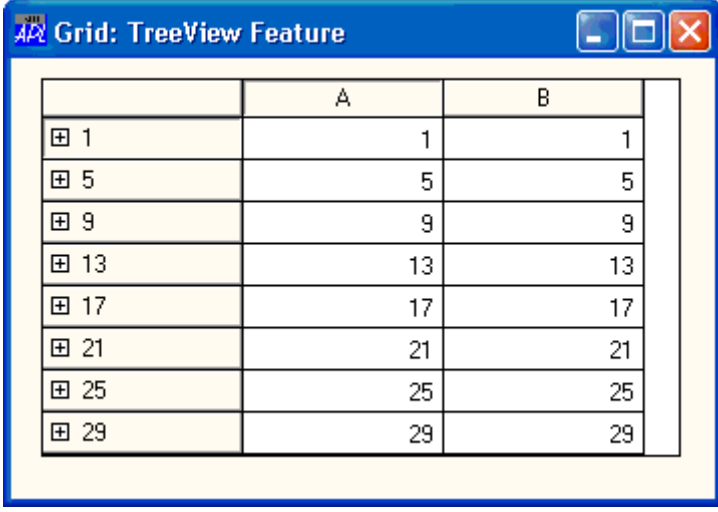
A value of 0 indicates that the corresponding row is a top-level row. A value of 1 indicates that the corresponding row is a child of the most recent row whose *RowTreeDepth* is 0; a value of 2 indicates that the corresponding row is a child of the most recent row whose *RowTreeDepth* is 1, and so forth.

When you set *RowTreeDepth*, the *Grid* is redrawn so that only rows with a *RowTreeDepth* of 0 are visible.

The *RowSetVisibleDepth* method can be used to make data visible to a specific depth.

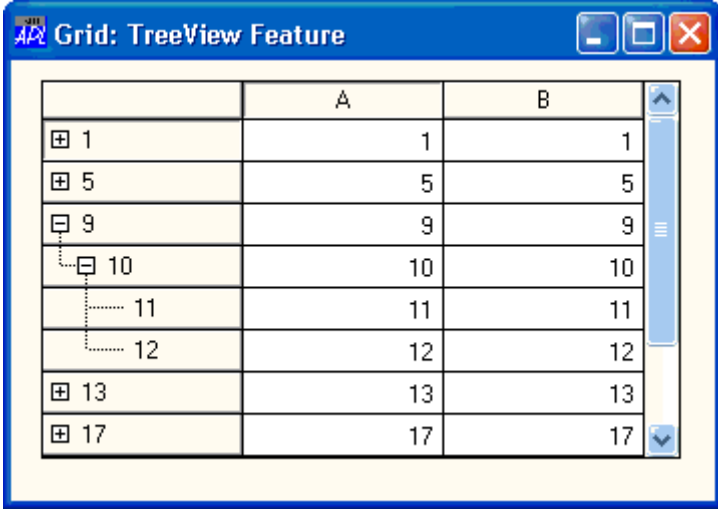
EXAMPLE

```
'F'WC'Form' 'Grid: TreeView Feature'
'F.G'WC'Grid'(30 2p2/t30)
F.G.RowTreeDepth←30p0 1 2 2
```



	A	B
⊕ 1	1	1
⊕ 5	5	5
⊕ 9	9	9
⊕ 13	13	13
⊕ 17	17	17
⊕ 21	21	21
⊕ 25	25	25
⊕ 29	29	29

The user can interact with the tree images to expand and contract rows of the *Grid*.



	A	B
⊕ 1	1	1
⊕ 5	5	5
⊖ 9	9	9
⊖ 10	10	10
⊖ 11	11	11
⊖ 12	12	12
⊕ 13	13	13
⊕ 17	17	17

3.261 RowTreeImages Property

Applies To: *Grid*

Description

The RowTreeImages property is a simple character vector or ref, or a vector of character vectors or refs, that specifies the names(s) of, or ref(s) to, Bitmap objects that are used to display the tree nodes for a *Grid* object.

Note that images in tree nodes are only displayed if *RowTreeStyle* is set to 'ImagesOnly', 'ImagesAndLines', or 'AllImagesAndLines'.

If *RowTreeImages* is not specified default images are used.

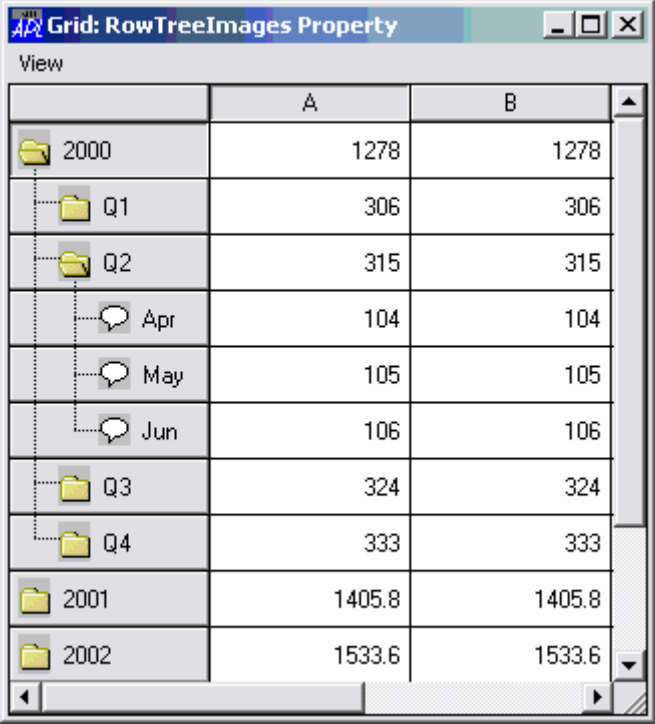
The Bitmap specified by the 1st element of *RowTreeImages* is used to display *unopened* nodes.

The Bitmap specified by the 2nd element of *RowTreeImages* is used to display *opened* nodes.

The Bitmap specified by the 3rd element of *RowTreeImages* is used to display nodes *without children*.

EXAMPLE

```
'Closed' <WC'Bitmap' 'Folder.bmp'
'Open' <WC'Bitmap' 'FolderOpen.bmp'
'Item' <WC'Bitmap' 'Ideas'
F.G.RowTreeStyle < 'AllImagesAndLines'
F.G.RowTreeImages < 'Closed' 'Open' 'Item'
```



View	A	B
2000	1278	1278
Q1	306	306
Q2	315	315
Apr	104	104
May	105	105
Jun	106	106
Q3	324	324
Q4	333	333
2001	1405.8	1405.8
2002	1533.6	1533.6

3.262 RowTreeStyle Property

Applies To: *Grid*

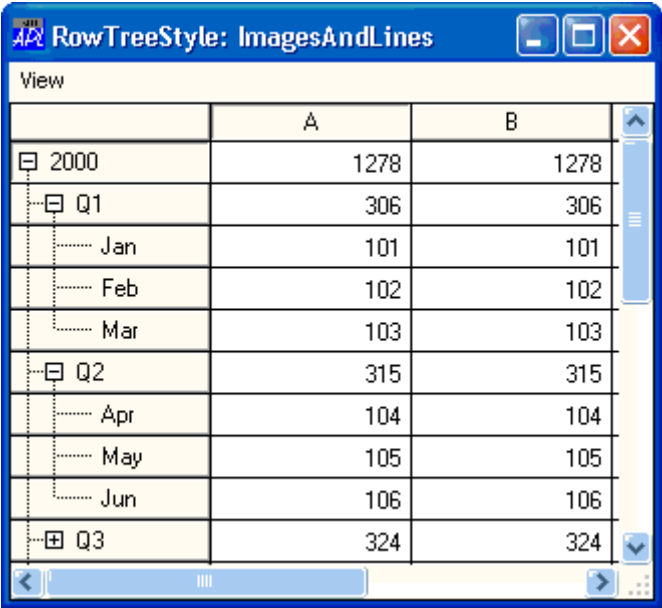
Description

RowTreeStyle specifies the visible attributes of the tree displayed in the Row titles of a *Grid*.

The value of the RowTreeStyle property is a character vector chosen from the following :

'LinesOnly '	Only the lines of the tree structure are drawn.
'ImagesOnly '	Only the images of nodes with children are drawn.
'ImagesAndLines '	Both lines and images for nodes with children are drawn.
'AllImagesOnly '	Images for all nodes are drawn.
'AllImagesAndLines '	Both lines and images for all nodes are drawn.

The default value, ' ImagesAndLines ', is illustrated in the first picture below. Other values are displayed in subsequent pictures.



F.G.RowTreeStyle←'LinesOnly '

RowTreeStyle: LinesOnly

View

	A	B
2000	1278	1278
Q1	306	306
Jan	101	101
Feb	102	102
Mar	103	103
Q2	315	315
Apr	104	104
May	105	105
Jun	106	106
Q3	324	324

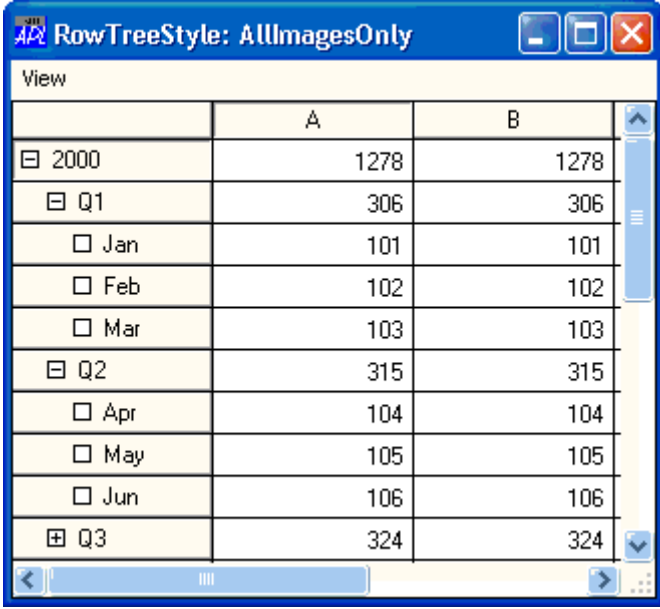
```
f.g.RowTreeStyle←'ImagesOnly'
```

RowTreeStyle: ImagesOnly

View

	A	B
2000	1278	1278
Q1	306	306
Jan	101	101
Feb	102	102
Mar	103	103
Q2	315	315
Apr	104	104
May	105	105
Jun	106	106
Q3	324	324

```
f.g.RowTreeStyle←'AllImagesOnly'
```

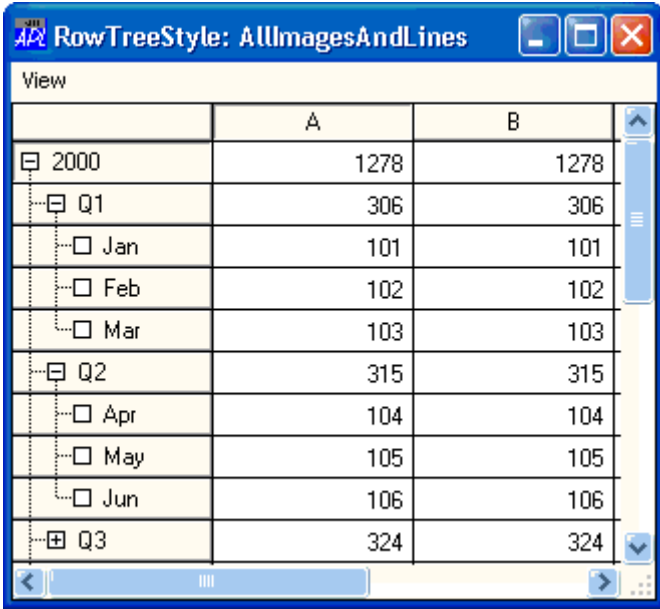


RowTreeStyle: AllImagesOnly

View

	A	B
2000	1278	1278
Q1	306	306
Jan	101	101
Feb	102	102
Mar	103	103
Q2	315	315
Apr	104	104
May	105	105
Jun	106	106
Q3	324	324

```
f.g.RowTreeStyle←'AllImagesAndLines'
```



RowTreeStyle: AllImagesAndLines

View

	A	B
2000	1278	1278
Q1	306	306
Jan	101	101
Feb	102	102
Mar	103	103
Q2	315	315
Apr	104	104
May	105	105
Jun	106	106
Q3	324	324

3.263 Rows Property

Applies To: *Combo*, *ComboEx*

Description

For *Combo* objects with *Style* 'Drop' or 'DropEdit' this property determines the number of rows displayed in the drop-down listbox when it is displayed. Note that the height of the edit field of a *Combo* of this type is dependent only upon the size of the font in use, and cannot otherwise be changed.

Rows is a "read-only" property for a *Combo* with *Style* 'Simple' and an attempt to set it in a *Combo* of this type with *Dyalog APL Language: Wc* or *Dyalog APL Language: Ws* will generate a NONCE ERROR. Instead, the overall height of a Simple *Combo* is determined by the first element of the *Size* property.

3.264 RunMode Property

Applies To: *OLEServer*

Description

This property specifies the way in which an *OLEServer* object serves multiple clients.

RunMode is a character vector and may be 'MultiUse' (the default), 'SingleUse' or 'RunningObject'.

If RunMode is 'MultiUse', OLE will load a single copy of Dyalog APL and the appropriate workspace into memory. All OLE client processes will communicate with the same Dyalog APL session.

Note that in this case, each OLE client is actually connected to a separate *instance* of the corresponding APL namespace. That is to say, each client will appear to have its own private copy of the namespace. However, the individual functions and variables in the namespace are not physically copied until they are changed. This means that, in general, OLE clients will share APL functions but have private copies of the namespace variables. However, please remember that global objects in the workspace or in other namespaces are not *instanced* and will effectively be shared by all clients although they are not directly accessible to them.

If RunMode is 'SingleUse', OLE will load a separate copy of Dyalog and a separate copy of the appropriate workspace into memory for each OLE client. Each OLE client operates directly on the namespace associated with the object and not an *instance* of it.

If RunMode is 'RunningObject', OLE will load a single copy of Dyalog and the appropriate workspace into memory. All OLE client processes will communicate with

the same Dyalog session and indeed with the same namespace. The namespace is not *instanced* and all objects, including exported variables, are shared by all clients.

3.265 ScrollOpposite Property

Applies To: *TabControl*

Description

The ScrollOpposite property specifies that unneeded tabs scroll to the opposite side of a *TabControl*, when a tab is selected.

ScrollOpposite is a single number with the value 0 (normal scrolling) or 1 (scrolling to the opposite side); the default is 0.

The picture below illustrates a *TabControl* with ScrollOpposite set to 1, after the user has clicked *Third Tab*.



Setting ScrollOpposite to 1 implies that *MultiLine* is also 1. If you set ScrollOpposite to 1 in a `⌈WC` statement, the *MultiLine* property will automatically be set to 1, even if you try to set *MultiLine* to 0 in the same statement. If you subsequently change *MultiLine* back to 0 using `⌈WS`, this will work, but the effect is not useful and it is not supported.

3.266 SelDate Property

Applies To: *Calendar*

Description

The SelDate property identifies the range of dates that is currently selected in a *Calendar* object.

SelDate is a 2-element integer vector of *IDN* values that identifies the first and last dates that are currently selected.

3.267 SellmageIndex Property

Applies To: *ComboEx*, *TreeView*

Description

The SellmageIndex property determines which bitmapped images in an *ImageList* correspond to items in a *TreeView* object when the item is *selected*. It is an integer vector whose length is the same as the number of items in the object and is $\square$ IO dependent.

See also *ImageIndex*

3.268 Selltems Property

Applies To: *Combo*, *ComboEx*, *Grid*, *List*, *ListView*, *TreeView*

Description

This property determines which (if any) of the items in an object are currently selected and highlighted. Except in a *Grid*, it is a Boolean vector with one element per item in the list. A value of 1 means "selected"; 0 means "not selected".

This property is used after a *Select* event to identify which item has been chosen. In a *Combo* or a *List* with *Style* 'single' it will contain only a single 1.

Selltems should also be used to pre-set the contents of the edit field in a *Combo* box with *Style* 'Drop'. In *Combo* boxes with *Style* 'Simple' or 'DropEdit', the contents of the edit field may also be specified by the *Text* property. If you specify both, the value of *Text* takes precedence.

In a *Grid*, Selltems is a 2-element integer vector. The first element identifies the row and column coordinates respectively of the first cell(s) in the selected block(s) and the second, the row and column coordinates of the last cell(s) in the selected block(s). If the *CellSelect* property allows only a single block of cells to be selected, each set of coordinates is a 2-element vector. If the *CellSelect* property permits more than one block of cells to be selected, each set of coordinates is a 2-column matrix with one row per selected block, whose columns identify the first row and column, and the last row and column respectively of each selected block.

3.269 SelRange Property

Applies To: *TrackBar*

Description

The SelRange property specifies the selected range in a TrackBar which has Style 'Selection'. It is a 2-element numeric vector.

3.270 SelText Property

Applies To: *ButtonEdit, Combo, ComboEx, Edit, RichEdit*

Description

This property determines or identifies the portion of text in an object that is currently selected and highlighted. It can be used to pre-select all or part of the text to be replaced or deleted when the user starts typing. It can also be used to query the area of text that the user has highlighted. This can be useful if you want to implement your own cut/paste/replace features.

SelText is always a 2-element integer vector. If the field contents (defined by the *Text* property) is a vector, SelText is simple. Its first element is the index of the first selected character and its second element is 1 + the index of the last selected character. The length of the selected string is therefore obtained by subtracting the first element from the second.

If there are no characters selected, the two elements are equal and specify the current position of the input cursor.

If the contents is a vector of vectors or a matrix, each element of SelText is a 2-element vector. The first item in each of the elements indexes the vector (in a vector of vectors) or row (in a matrix). The second item in each element indexes the position of the character in the vector or along the row. Again, the value reported for the last character in the selected string is 1 + its index.

3.271 SelectionBorderWidth Property

Applies To: *Grid*

Description

The `SelectionBorderWidth` property specifies the width of the border that is drawn around the currently selected block of cells. It is expressed in pixels.

3.272 SelectionColor Property

Applies To: *Grid*

Description

The `SelectionColor` property specifies the colour used to highlight the currently selected block of cells and, if *HighlightHeaders* is 1, the corresponding row and column headings. See also *SelectionColorAlpha*.

3.273 SelectionColorAlpha Property

Applies To: *Grid*

Description

The `SelectionColorAlpha` property is a 2-element integer vector that specifies the degree of transparency or shade of the colour that is used to highlight the currently selected block of cells in a *Grid*. See *SelectionColor*.

The first element refers to the shade to be used when the *Grid* has the input focus; the second to when it doesn't. Each element is an integer in the range 0 (invisibly light) to 255 (fully dark).

3.274 ServerVersion Property

Applies To: *OLEServer*

Description

This property specifies the version number of an *OLEServer* object.

It is a 2-element integer vector that specifies the major and minor version numbers respectively.

The default value of `ServerVersion` is (1 0).

3.275 ShowCaptions Property

Applies To: *ToolControl*

Description

The ShowCaptions property specifies whether or not the captions of individual *ToolButton* objects are drawn. ShowCaptions is a property of the parent *ToolControl* object.

ShowCaptions is a single number with the value 0 (*ToolButton* captions are not shown) or 1 (*ToolButton* captions **are** shown); the default is 1

ShowCaptions allows you to toggle end-user preferences for the display of *ToolButton* captions, without having to set/clear individual captions one by one.

3.276 ShowCueWhenFocused Property

Applies To: *ButtonEdit*, *Edit*

Description

This Boolean property specifies whether or not the text specified by the *Cue* property should be displayed once the user has tabbed into or clicked on the empty input field (and thus given it the focus). For an *Edit* object it applies only if the *Style* of the *Edit* object is 'Single'.

3.277 ShowDropDown Property

Applies To: *ColorButton*, *ToolControl*

Description

The ShowDropDown property specifies whether or not a drop-down menu symbol is drawn in a *ColorButton* or alongside *ToolButton* objects which have *Style* 'DropDown'.

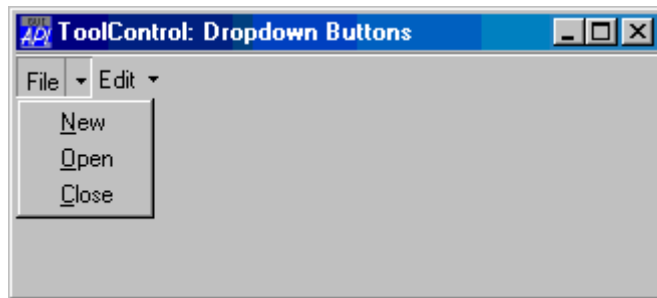
ShowDropDown is a single number with the value 0 (drop-downs captions are not shown) or 1 (drop-downs **are** shown); the default is 1.

ShowDropDown also affects the behaviour of *ToolButton* objects which have *Style* 'DropDown'.

If the `ShowDropDown` property of the parent *ToolControl* is 0, clicking the *ToolButton* causes the popup menu to appear. In this case, the *ToolButton* itself does not itself generate a `Select` event; you must rely on the user selecting a *MenuItem* to specify a particular action.

If the `ShowDropDown` property of the parent *ToolControl* is 1, clicking the dropdown button causes the popup menu to appear; clicking the *ToolButton* itself generates a `Select` event, but does not display the popup menu.

The following picture illustrates a *ToolControl* with `ShowDropDown` set to 1:



3.278 ShowInput Property

Applies To: *Grid*

Description

This property specifies whether or not the cells in a *Grid* are displayed using their associated input objects.

The `ShowInput` property is either a single Boolean value that applies to all the cells in a *Grid*, or it is a vector whose elements are mapped to individual cells via the *CellTypes* property. A value of 0 means that the corresponding cell is displayed normally. A value of 1 indicates that the cell is displayed using its associated input object, as it is when it is the current cell. `ShowInput` is relevant to cells displayed using *Combo* and *Button* objects.

The example below illustrates the appearance of a *Grid* in which `ShowInput` is set to 0 for the Job Title column and 1 for the Region and Permanent columns.

Employee Database					
Surname	Job Title	Region		Salary	Permanent
Brown	Manager	South	↓	\$64000.00	☒
Jones	Project Leader	South	↓	\$43250.00	☒
Green	Consultant	South	↓	\$45000.00	☐
Black	Programmer	East	↓	\$30000.00	☒
White	Assistant	Central	↓	\$40000.00	☐

The appearance of the same *Grid* but with *ShowInput* set to 0 throughout is illustrated below:

Employee Database					
Surname	Job Title	Region		Salary	Permanent
Brown	Manager	South		\$64000.00	1
Jones	Project Leader	South		\$43250.00	1
Green	Consultant	South	↓	\$45000.00	
Black	Programmer	East		\$30000.00	1
White	Assistant	Central		\$40000.00	

3.279 ShowSession Property

Applies To: *OLEServer*

Description

This property specifies whether or not the APL Session window is displayed when an *OLEServer* object is started by an OLE client.

Its default value is 0 (hide Session).

Note that if *RunMode* is 'MultiUse', you may not in any way access the *instances* of the object that are being controlled by the client applications, even if only a single client is connected.

3.280 ShowThumb Property

Applies To: *TrackBar*

Description

The ShowThumb property specifies whether or not the thumb in a *TrackBar* object is visible. It is Boolean with a default value of 1 and it may be toggled on and off using `WS`.

3.281 SingleClickExpand Property

Applies To: *TreeView*

Description

The SingleClickExpand property specifies whether or not an item in a *TreeView* control is expanded when the user selects the item.

SingleClickExpand is a single number with the value 0 (the user must select the *expand icon* to cause the item to expand) or 1 (the item is expanded when the *text* of the item is selected); the default is 0.

3.282 Size Property

Applies To

Description

This is a 2-element numeric vector specifying the height and width of the object.

For the *Bitmap* object, Size is set and reported in pixels. Setting the Size of a *Bitmap* causes it to be scaled (up or down).

For all other objects, Size is reported and set in units defined by the *Coord* property and, if *Coord* is 'User', the *XRange* and *YRange* properties of the object's parent.

For the *Root* object, if *Coord* is 'Prop' the value of Size is (100,100). If *Coord* is 'Pixel' the value of Size reports the number of pixels on the screen.

For a *Form* or *SubForm*, the Size property defines the area within the object, and excludes its title bar, menu bar and border if these are present.

For a *Combo* object with a "drop-down" list, the first element of *Size* (height) is ignored. The height of the edit field is determined by the height of the font, while the size of the list box is determined by the *Rows* property.

Otherwise the *Size* property defines the total size of the object, including borders, edges etc.

When specifying *Size*, you can set the height or width to a default value ($\square Wc$) or leave it unchanged ($\square Ws$) by giving the corresponding element a value of $\emptyset$.

3.283 Sizeable Property

Applies To

Description

This property determines whether or not an object can be directly resized by the user once it has been created by *Dyalog APL Language: Wc*.

It is a single number with the value 0 (the object cannot be resized by the user) or 1 (the object may be resized by the user). The default is 1.

For a *Form*, *HTMLRenderer* or *SubForm*, the *Sizeable* property may only be set by *Dyalog APL Language: Wc* and cannot subsequently be altered using *Dyalog APL Language: Ws*. An attempt to do so generates a `NONCE ERROR`. For a *Form* or *HTMLRenderer*, the default value is 1 and the object occupies a standard resizable window with a border. Note that the value of *Sizeable* is independent of the values of the *MaxButton* and *MinButton* properties, so that a *Form* or *HTMLRenderer* with *MaxButton* 1 can be maximised even though its *Sizeable* property is 0.

For other objects, the default value of the *Sizeable* property is 0. However, setting it to 1 (which may be done dynamically using *Dyalog APL Language: Ws*) allows the user to resize it with the mouse.

In all these cases, when the user resizes an object, the object will generate a *Configure* (31) event.

Sizeable also applies to the *Locator* object. In this case, a value of 1 implies "rubberbanding" and a value of 0 means "no rubberbanding". See *Locator* object for further details.

3.284 SocketNumber Property

Applies To: *TCPSocket*

Description

The `SocketNumber` property is an integer whose value is the Window handle of the socket attached to the *TCPSocket* object and is generally a read-only property.

The only time that `SocketNumber` may be specified is when a server replicates (clones) a listening socket to which a client has just connected.

3.285 SocketType Property

Applies To: *TCPSocket*

Description

The `SocketType` property is a character vector that specifies the type of the TCP/IP socket. This is either *Stream* (which is the default), or *UDP*.

`SocketType` must be defined when the object is created and may not be set or changed using `□WS`.

For two Dyalog applications to communicate, their *TCPSocket* objects must have the same `SocketType`.

3.286 SortItems Property

Applies To: *List*

Description

The `SortItems` property specifies whether or not the items in a *List* object are sorted.

It is Boolean with a default value of 0. If `SortItems` is 1, the items are automatically sorted in alphabetical order and the object provides word recognition capabilities for selecting an item from the keyboard.

Note that the value of the *Items* property reflects the order of the items displayed in the *List* object rather than the order in the array that was used to assign it.

This property may only be initialised when the object is created and cannot subsequently be changed.

3.287 SplitObj1 Property

Applies To: *Splitter*

Description

The SplitObj1 property specifies the name of, or ref to, one of up to two objects managed by a *Splitter* object. The object must be one of the following types:

Button	Calendar	Combo	Edit	Grid	Group
Label	List	ListView	MDIClient	ProgressBar	RichEdit
Scroll	Spinner	Static	StatusBar	SubForm	TabBar
TabControl	ToolBar	TrackBar	TreeView	UpDown	

If the *Style* property of the *Splitter* is 'Vert', the object specified by SplitObj1 is positioned at (0 0) and sized to occupy the space in its parent to the left of the *Splitter*, with the *Splitter* itself attached to its right edge.

If the *Style* property of the *Splitter* is 'Horz', the object specified by SplitObj1 is positioned at (0 0) and sized to occupy the space in its parent above the *Splitter*, with the *Splitter* itself attached to its bottom edge.

If SplitObj1 is empty, the *Splitter* manages the single object specified by *SplitObj2* and the space to the left or above the *Splitter* is empty or controlled by another *Splitter*.

3.288 SplitObj2 Property

Applies To: *Splitter*

Description

The SplitObj2 property specifies the name of, or ref to, one of up to two objects managed by a *Splitter* object. The object must be one of the following types:

Button	Calendar	Combo	Edit	Grid	Group
Label	List	ListView	MDIClient	ProgressBar	RichEdit
Scroll	Spinner	Static	StatusBar	SubForm	TabBar
TabControl	ToolBar	TrackBar	TreeView	UpDown	

If the *Style* property of the *Splitter* is 'Vert', the object specified by *SplitObj2* is initially positioned at (0 x), where x is half the width of the parent plus the Size of the *Splitter*, and sized to occupy the space in its parent to the right of the *Splitter*, with the *Splitter* itself attached to its left edge.

If the *Style* property of the *Splitter* is 'Horz', the object specified by *SplitObj2* is initially positioned at (y 0), where y is half the height of the parent plus the Size of the *Splitter*, and sized to occupy the space in its parent below the *Splitter*, with the *Splitter* itself attached to its top edge.

If *SplitObj2* is empty, the *Splitter* manages the single object specified by *SplitObj1* and the space to the right or below the *Splitter* is empty or controlled by a second *Splitter*.

3.289 Start Property

Applies To: *Circle, Ellipse*

Description

This property specifies one or more start-angles for an arc, pie-slice, or chord of a circle or ellipse. It may be used in conjunction with *End* which specifies end angles. Angles are measured counter-clockwise from the x-axis at the centre of the object.

If a single arc is being drawn, *Start* is a single number that specifies the start angle of the arc in radians (0 ≤ < 2). If multiple arcs are being drawn, *Start* is either a single number as before (the start angle for several concentric arcs) or a numeric vector with one element per arc.

If *End* is not specified, the default value of *Start* is 0. Otherwise, the default value of *Start* is (0, -1 ↓ + \End).

3.290 StartIn Property

Applies To: *BrowseBox*

Description

The `StartIn` property is a character string that specifies the start point and root for a `BrowseBox` object.

Only the specified folder and its subfolders appear in the dialog box. The user cannot browse higher in the folder architecture than this folder.

The default value for `StartIn` is an empty vector which means that the root of the browse dialog is the desktop.

3.291 State Property

Applies To: *Button, Form, SubForm, TabButton, ToolButton*

Description

This property determines the state of a *Button*, *TabButton*, *ToolButton*, *Form*, or *SubForm*. It is a single number with the value 0 (the default), 1, or 2 (*Form* and *SubForm*).

If the *Style* property is 'Push', a State of 0 means that the pushbutton is displayed normally (out). If its State is 1, the pushbutton is displayed depressed (in).

If the *Style* property is 'Radio' or 'Check', 0 means "not selected" and 1 means "selected". Note that only one of a group of buttons with *Style* 'Radio' that share the same parent may have State 1. Setting State to 1 automatically deselects all the others in the group.

For a *Form* or *SubForm*, a value of State of 0 means that the *Form* is currently displayed in its "normal" state. 1 means that the *Form* is currently minimised (displayed as an icon). The value 2 indicates that the *Form* is maximised and displayed full-screen. The State of a *Form* can be changed using *Dyalog APL Language: Ws*.

3.292 Step Property

Applies To

Description

This property determines the size of changes reported when the user clicks a scroll arrow (small change) or clicks on the body of the scrollbar (large change). The object's *Thumb* property increases or decreases by this amount.

For a *Scroll* object, *Step* is a 2-element numeric vector whose first element specifies the value of the "small change" and whose second element specifies the value of the "large change".

For a *Form* or *SubForm*, *Step* is a 4-element numeric vector. The first two elements refer to the *Form*'s vertical scrollbar and the second two elements refer to the *Form*'s horizontal scrollbar.

For the above objects, values of *Step* must be between 1 and the value of the *Range* property.

For a *Locator* object, *Step* is a 2-element integer vector (default value 1 1) that specifies the increments (in pixels) by which the size or position of the *Locator* changes in the Y and X directions respectively as the user moves the *Locator*.

3.293 Style Property

Applies To

Description

This property determines a particular style of object within the general category of *Type*.

It is a character vector whose value depends upon the type of object.

For a *Button*, *Style* may be 'Push', 'Radio', 'Check', 'Toggle', 'CommandLink' or 'Split'.

'Push' specifies that the button appears and behaves like a pushbutton (sometimes also called a command button).

'Radio' means that the button is displayed as a small circle accompanied by a description. When the button is selected, the circle is filled in. In a group of buttons with *Style* 'Radio' that share the same parent, only one of them may be selected. This style of button is generally known as a "radio-button" or an "option button".

'Check' or 'Toggle' means that the button is displayed as a small box accompanied by a description. When the button is selected a cross appears in the box. This style of button is known as a "check-box".

'CommandLink' means that the button has an icon displayed to the left of its *Caption*, the appearance of which is controlled by the *Elevated* property. **Note that this feature only applies if Native Look and Feel

is enabled.**

'Split' specifies a 'Push' button with an additional drop-down button, similar to that provided by a *Combo* object. **Note that this feature only applies if Native Look and Feel

is enabled.**

For a *Calendar* object, The Style property may be either 'Single' (the default) or 'Multi'. If Style is 'Single', the user may select a single date. If Style is 'Multi', the user may select a contiguous range of dates.

For a *Combo* or *ComboEx* object, Style may be 'Simple' 'DropEdit' or 'Drop' (the default). 'Simple' specifies a simple combo box in which the associated list box is displayed at all times. The other two styles provide list boxes which "drop down" when the user clicks on a symbol displayed to the right of the *Combo*'s edit field. A 'DropEdit' Style allows the user to type (anything) in the edit field. A 'Drop' Style forces the contents of the edit field to be either empty or one of the choices specified by *Items*.

For a *DateTimePicker*, Style may be either 'Combo' (the default) or 'UpDown'.

For an *Edit* object, Style may be 'Single' or 'Multi'. If Style is 'Single' the object displays only a single line of text and the user may not enter any more lines. If the Style is 'Multi' the number of lines displayed is governed by the *Rows* or *Size* property and the user may insert, add or delete lines as desired.

For *FileBox*, *List*, and *ListView* objects, Style may be 'Single' or 'Multi'. If the Style is 'Single' only one file or item can be selected. If Style is 'Multi', several files or items can be selected.

For an *Icon*, Style may be 'Large' (the default) or 'Small' and specifies the size of the icon (32x32 or 16x16) to be loaded from a file.

For a *Locator*, Style may be 'Point', 'Rect' (the default), 'Line' or 'Ellipse'. It specifies the shape that is drawn as the user moves the mouse.

For a *MenuItem*, Style may be 'Check' (the default) or 'Radio'. The latter specifies that within a contiguous block of such *MenuItems*, only one may have *Checked* set to 1. Setting *Checked* to 1 on any item in that group automatically sets *Checked* to 0 on the others. A radio style *MenuItem* that is checked has a small radio dot drawn to the left of its Caption.

For a *MsgBox*, the *Style* property determines the type of icon which is displayed in it. This is a character vector with one of the following values:

'Msg'	no icon (the default)
'Info'	information message icon
'Query'	query (question) icon
'Warn'	warning icon
'Error'	critical error icon

For a *Static* object, *Style* defines its appearance, and may be one of:

'BlackFrame'	'BlackBox'
'GreyFrame' or 'GrayFrame'	'GreyBox' or 'GrayBox'
'WhiteFrame'	'WhiteBox'

A *StatusField* may be used to monitor the state of a key on the keyboard. If so, its *Style* property determines the key it monitors and may be one of the following:

CapsLock	Monitors state of Caps Lock key
ScrollLock	Monitors state of Scroll Lock key
NumLock	Monitors state of Num Lock key
KeyMode	Monitors the keyboard mode (APL/ASCII)
InsRep	Monitors the state of the Insert key

For a *Splitter*, the *Style* property specifies the orientation of the *Splitter* and may be 'Vert' (the default) or 'Horz'.

For a *TabControl*, the *Style* property determines the appearance of its *TabButton* children, and may be 'Tabs' (the default), 'Buttons' or 'FlatButton'.

For a *TCPSocket*, *Style* is a character vector that specifies the type of data transmitted or received by the socket; it may be 'Char', 'Raw', or 'APL'. The value APL is valid only if the *SocketType* is 'Stream'.

For a *ToolButton*, the *Style* property specifies the behaviour of the button and may be 'Push' (the default), 'Check', 'Radio', 'DropDown', or 'Separator'.

For a *ToolControl*, the *Style* property determines the appearance of its *ToolButton* children and may be 'Buttons', 'FlatButton' (the default), 'List' or 'FlatList'.

For a *TrackBar*, the *Style* property determines the appearance and behaviour of the *TrackBar* and may be 'Standard' (the default) or 'Selection'.

3.294 SysMenu Property

Applies To: *Form*, *HTMLRenderer*, *SubForm*

Description

This property determines whether or not a *Form* or *SubForm* has a "System Menu " box in the top-left corner of its border. Pressing the left mouse button in this box brings up the standard window control menu for the *Form*. Double clicking the box closes the *Form*.

SysMenu is a single number with the value 0 (no System menu box) or 1 (System Menu box is provided). The default is 1.

If any of the *SysMenu*, *MaxButton*, *MinButton* and *Moveable* properties are set to 1, the *Form* or *SubForm* has a title bar.

3.295 TabFocus Property

Applies To: *TabControl*

Description

The *TabFocus* property specifies the focus behaviour for the *TabControl* object and may

TabFocus is a character vector that may be 'Normal' (the default), 'Never' or 'ButtonDown'.

If *TabFocus* is 'Normal', the tabs or buttons in a *TabControl* do not immediately receive the input focus when clicked, but only when clicked a second time. This means that, normally, when the user circulates through the tabs, the input focus will be given to the appropriate control in the associated *SubForm*. However, if the user clicks twice in succession on the same tab or button, the *TabControl* itself will receive the input focus.

If *TabFocus* is 'ButtonDown', the tabs or buttons in a *TabControl* receive the input focus when clicked.

If `TabFocus` is 'Never', the tabs or buttons in a *TabControl* *never* receive the input focus. This allows the user to circulate through a set of tabbed SubForms without ever losing the input focus to the *TabControl* itself.

3.296 TabIndex Property

Applies To

Description

The `TabIndex` property reports the $\square IO$ -dependent relative position of a child object within the list of child objects owned by its parent. If N is the number of children owned by an object, `TabIndex` is an integer between $\square IO$ and $(N - \sim \square IO)$. The sequence of objects in this list is also used as the tabbing sequence, that is, if the input focus is on the first child in the list, pressing `Tab` moves the input focus to the next child in the list.

When you create a child object, it is inserted in the list at the position specified by its `TabIndex` property. If `TabIndex` is omitted, it is appended to the end of the list.

If you subsequently change `TabIndex`, the object is moved to the corresponding position in the list.

Naturally, if you specify a value of `TabIndex` that is greater than the number of existing children, the object is inserted at or moved to the end of the list.

3.297 TabJustify Property

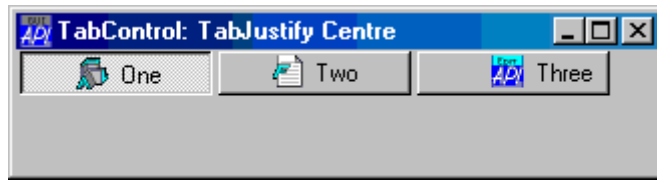
Applies To: TabControl

Description

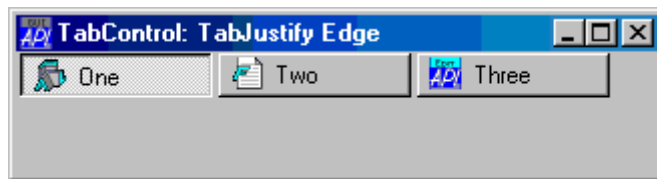
The `TabJustify` property specifies, the positions at which the picture and caption are drawn within each tab or button implemented by a *TabButton* in a *TabControl* object.

`TabJustify` is a character vector that may be 'Centre', 'Edge', or 'IconEdge'. Its default value is 'Centre'.

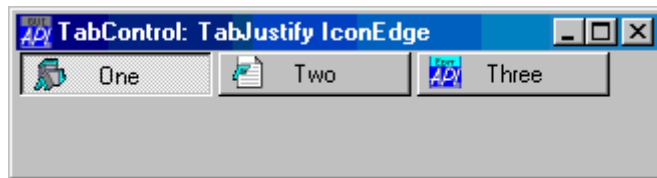
If `TabJustify` is 'Centre', the picture and caption are arranged in the centre of the *TabButton*.



If `TabJustify` is 'Edge', the picture and caption are together aligned to the appropriate edge of the `TabButton` according to the value of `Align`.



If `TabJustify` is set to 'IconEdge', the caption is drawn centrally and only picture is aligned to the edge.



`TabJustify` is only honoured if fixed size tabs are in effect.

3.298 TabObj Property

Applies To: *SubForm*, *TabBar*, *TabBtn*, *TabButton*, *TabControl*

Description

`TabObj` is a ref or a character vector.

`TabObj` associates a *SubForm* with a *TabBtn* or a *TabButton* object. Selecting the associated *TabBtn* or *TabButton* causes the *SubForm* to be given the input focus.

For a *SubForm*, it specifies the name of, or ref to, a *TabBtn* or *TabButton* object that is to be associated with the *SubForm*. When referenced or queried using `□WG`, `TabObj` returns a name if it was specified by a name, or a ref if it was specified by a ref.

For *TabBtn* and *TabButton* objects, `TabObj` is a read-only property that contains a ref to the associated *SubForm*.

For a *TabBar* or *TabControl*, *TabObj* is a read-only property that contains a ref to the currently selected *TabBtn* or *TabButton*.

3.299 TabSize Property

Applies To: *TabControl*

Description

The *TabSize* property specifies the size of fixed size tabs or buttons in a *TabControl* object.

By default, the size of the tabs may vary from one to another. Fixed size tabs may be obtained by setting the *TabSize* property.

TabSize is a 2-element numeric vector that specifies the height and width of the tab. The first element of *TabSize* may be set to 0 which means "default height".

To obtain fixed sized tabs with *MultiLine* set to 1, you must however also set the *Justify* property to 'None'.

If *MultiLine* is 1 and *Justify* is 'Right', *TabSize* is ignored.

3.300 Target Property

Applies To: *BrowseBox*

Description

The *Target* property is a read-only character string that specifies the chosen folder or other resource selected by the user in a *BrowseBox* object.

If the *BrowseFor* is 'Directory', *Target* will contain a directory name followed by the character "\". Otherwise, *Target* just contains the name.

3.301 TargetState Property

Applies To: *TCPSocket*

Description

The `TargetState` property reflects the intended final state of a `TCPSocket` object. Its possible values are as follows:

Stream	UDP
Client	Open
Server	Bound
Closed	Closed

Setting `TargetState` to `Closed` is the recommended way to close a socket. It informs APL that you want the socket to be closed, but only when it is safe to do so. When all the data has been sent, the `TCPSocket` will generate a `TCPClose` event which, unless a callback function decides otherwise, will cause the `TCPSocket` object to disappear.

To control socket closure, you may execute the following steps:

1. Set `TargetState` to `Closed`
2. **Either:**
 1. continue processing **or**
 2. wait (using `□□□`) for the `TCPSocket` to disappear **or**
 3. wait (using `□□□`) for the `TCPClose` event

3.302 Text Property

Applies To

Description

This property is associated with the text contents of an object and is a character array.

In a *ButtonEdit*, *Combo*, *StatusField*, *Spinner*, or a single-line *Edit* object, `Text` may be a simple scalar or a simple vector.

In a *RichEdit*, a multi-line *Edit* field or in a *MsgBox*, the value of `Text` may also be a simple matrix, or a vector of vectors. If so, "new-line" characters are appended to each row of the matrix, or to each vector in a vector of vectors, before being displayed. The user may insert or add a "new-line" character in a multi-line *Edit* by pressing Ctrl-Enter (Enter itself is used to press *Buttons*).

Note that if word-wrapping is in effect in a multi-line *Edit* object, the structure of `Text` does not correspond to the lines displayed.

In a *Text* object, the value of the *Text* property may be a simple scalar, an enclosed vector or matrix, a simple vector, a simple matrix, or a vector of enclosed vectors or matrices.

In general, the value of *Text* returned by *Dyalog APL Language: Wg* has the same structure that was assigned to it by *Dyalog APL Language: Wc* or by the most recent call to *Dyalog APL Language: Ws*. New-Line characters entered by the users are removed.

You can copy text into the Windows *Clipboard* by using *Dyalog APL Language: Ws* to set *Text* for a *Clipboard* object. In this case you may specify a simple character scalar, vector or matrix, or a vector of character vectors. If you are retrieving data from the clipboard that has been stored by another application, *Text* will be either a character vector or a vector of character vectors.

The *Text* property of a *StatusField* is updated automatically if its *Style* property is set to monitor the status of a key.

3.303 TextSize Property

Applies To

Description

This property has been replaced by the *GetTextSize* method, which should be used instead. *TextSize* is retained only for compatibility with previous versions of Dyalog.

TextSize is a "read-only" property that reports the size of the bounding rectangle of a text item in a given font. The result is given in the co-ordinate system of the object in question. This property is useful for positioning *Text* objects.

When you query *TextSize* you give the text item in whose size you are interested and, optionally, the name of a *Font* object. The text item may be a simple scalar, a vector or a matrix. If the *Font* is omitted, the result is given using the current font for the object in question. When you query *TextSize* on its own, you must enclose the argument to *Dyalog APL Language: Wg*. This is because APL would otherwise not be able to distinguish between the text string and font name associated with '*TextSize*' and other properties with the same name as these items.

EXAMPLES

```

    '.' □WG c'TextSize' 'Hello World'
2.6666666746 9.625

    'FNT1' □WC 'FontObj' 'Arial' 72
    '.' □WG c'TextSize' 'Hello World' 'FNT1'
12 41.875

    '.' □WS 'Coord' 'Pixel'
    '.' □WG ('TextSize' (3 11p'Hello World')) 'Coord'
39 55 Pixel

```

3.304 Thumb Property

Applies To

Description

This property determines and reports the position of the *thumb* in an object.

For a *Scroll* object, the value of *Thumb* is a single integer whose minimum value is 1 and whose maximum value is defined by the *Range* property.

For *ProgressBar*, *Spinner*, *UpDown* and *TrackBar* objects, *Thumb* is a single numeric value in the range specified by the *Limits* property.

For a *Form* or *SubForm* object, *Thumb* is a 2-element vector whose elements refer to the position of the thumb in the object's own built-in vertical and horizontal scrollbars respectively.

For other objects, *Thumb* is a single numeric value in the range defined by the *Limits* property.

3.305 ThumbRect Property

Applies To: *TrackBar*

Description

ThumbRect is a *read-only* property that reports the position and size of the bounding rectangle of the thumb in a *TrackBar* object. It is a 4-element integer vector containing:

[1]	Vertical position of the top-left corner of the bounding rectangle
[2]	Horizontal position of the top-left corner of the bounding rectangle
[3]	Height of the bounding rectangle
[4]	Width of the bounding rectangle

3.306 TickAlign Property

Applies To: *TrackBar*

Description

TickAlign determines the position of the tick marks in a *TrackBar* object. For a horizontal *TrackBar*, TickAlign may be either 'Bottom' (the default), 'Top' or 'Both'. If TickAlign is 'Bottom', the ticks are drawn *below* the slider. If TickAlign is 'Top', the ticks are drawn above it. If TickAlign is 'Both', the ticks are drawn above and below.

For a vertical *TrackBar*, TickAlign may be either 'Right' (the default), 'Left', or 'Both' and similarly specifies to which side of the slider bar the ticks are drawn. Note that TickAlign may only be set when the *TrackBar* is created with `WC` and may not subsequently be altered using `WS`.

Note that ticks are not drawn if the value of *HasTicks* is 0

3.307 TickSpacing Property

Applies To: *TrackBar*

Description

The TickSpacing property specifies the spacing between each tick mark in a *TrackBar* object. It is an integer between 1 and the maximum value of the *TrackBar* which is defined by the 2nd element of the *Limits* property.

For example, if you set ('Limits' 10 50) and you specify ('TickSpacing' 10) you will obtain 5 ticks corresponding to the values 10, 20, 30, 40 and 50 along the slider bar.

3.308 Tip Property

Applies To

Description

The *Tip* property is a character vector or character matrix that specifies a "help" message which is to be displayed when the user positions the mouse pointer over the object. The *Tip* is displayed in a pop-up *TipField* object specified by the *TipObj* property.

3.309 TipObj Property

Applies To

Description

The *TipObj* property is a character vector or ref that specifies the name of, or ref to, a *TipField* object in which the "help" message defined by the *Tip* property is to be displayed. This message is displayed when the user positions the mouse pointer over the object.

Note that if *TipObj* is empty, its value is inherited from its parent. Thus setting *TipObj* on a *Form* defines the default *TipField* (and thus the default appearance of all *Tips*) for all the controls in that *Form*. Setting *TipObj* on *Root* defines the default *TipField* for the entire application.

3.310 TitleHeight Property

Applies To: Grid

Description

This property is a single number that specifies the height of the column titles displayed in a *Grid* object. It is expressed in the units specified by the *Coord* property of the *Grid*. The value 0 selects the default height.

3.311 TitleWidth Property

Applies To: Grid

Description

This property is a single number that specifies the width of the row titles displayed in a *Grid* object. It is expressed in the units specified by the *Coord* property of the *Grid*. The value 0 selects the default width.

3.312 Today Property

Applies To: *Calendar*, *DateTimePicker*

Description

The Today property is an *IDN* that specifies today's date in a *Calendar* or *DateTimePicker* object. Its default value is the current date that is set on your computer.

See also *CircleToday* and *HasToday* properties.

3.313 ToolboxBitmap Property

Applies To: *ActiveXControl*, *OCXClass*

Description

For an *ActiveXControl*, the ToolboxBitmap property is a character vector or ref that specifies the name of, or ref to, a *Bitmap* object that may be used by a host application to represent the *ActiveXControl* when its complete visual appearance is not required. For example, if you add an ActiveX control to the Microsoft Visual Basic development environment, its bitmap is added to the toolbox. The Bitmap should therefore be of an appropriate size, usually 24 x 24 pixels.

For an *OCXClass*, The ToolboxBitmap is a read-only property that reports a bitmap image associated with an OLE Control. This is intended for use by a GUI design tool. Its value is a 2-element vector. The first element is an integer matrix of pixel colours corresponding to the Bits property of a Bitmap object. The second element is a 3-column integer matrix specifying the colour map and corresponds to the CMap property of a Bitmap object.

Thus you can construct a Bitmap object directly from this property with an expression such as:

```
'BM' □ WC 'Bitmap' ' ', 'GAUGE' □ WG 'ToolboxBitmap'
```

where GAUGE is the name of an *OCXClass*.

3.314 TrackRect Property

Applies To: *TrackBar*

Description

`TrackRect` is a *read-only* property that reports the position and size of the bounding rectangle of the slider in a *TrackBar* object. It is a 4-element integer vector containing:

[1]	Vertical position of the top-left corner of the bounding rectangle
[2]	Horizontal position of the top-left corner of the bounding rectangle
[3]	Height of the bounding rectangle
[4]	Width of the bounding rectangle

3.315 Translate Property

Applies To

Description

This property specifies whether or not character data is to be translated.

Translate applies to the Classic Edition only. In the Unicode Edition, its value is ignored.

Translate is a character vector whose values may be `'Inherit'`, `'Translate'`, `'ANSI'` or `'None'`.

A value of `'Translate'` means that all character property values and event parameters are translated to and from `□AV` using the current output translation table (WIN.DOT).

`'None'` means that character data is passed between APL and the object with no translation.

If you set the value of the Translate property to `'ANSI'`, APL does not attempt to resolve characters as they are typed by the user via the Input Translate Table. Using Translate `'ANSI'` in combination with the appropriate value of *CharSet* and the corresponding National Language keyboard will permit users to enter strings in non-western languages.

`'Inherit'` means that the object inherits its translation from its parent.

The default value for the Root and Printer objects is `'Translate'`, and for most other objects it is `'Inherit'`.

3.316 Transparent Property

Applies To: *Animation, ButtonEdit, ToolControl*

Description

The Transparent property specifies whether or not a *ToolControl* is transparent.

Transparent is a single number with the value 0 (the default) or 1.

If Transparent is 1, the visual effect is as if the *ToolButtons* (and other controls owned by the *ToolControl*) were drawn directly on the parent Form as illustrated below.



3.317 Type Property

Applies To

Description

This property determines the type of an object. Its value is a character vector containing a valid object type. The Type property is set by *Dyalog APL Language: Wc* and reported by *Dyalog APL Language: Wg*, but may not be altered using *Dyalog APL Language: Ws*.

3.318 TypeLibFile Property

Applies To: *ActiveXControl, OLEServer*

Description

The TypeLibFile property is a read-only property that reports the name of the file in which the Type Library associated with a COM object is stored.

3.319 TypeLibID Property

Applies To: *ActiveXControl, OLEServer*

Description

The TypeLibID property is a read-only property that reports the value of the globally unique identifier (*GUID*) of the Type Library associated with a COM object.

3.320 TypeList Property

Applies To: *OCXClass*, *OLEClient*

Description

This property reports the names of all the special data types defined for a particular COM object. It is a vector of character vectors returned by `TypeInfo`. It may not be set using `TypeInfo` or `TypeInfo`. Further information about each data type may be obtained using *GetTypeInfo*.

Note that TypeList reports **all** of the data type names recorded in the .OCX file associated with the COM object. If several COM objects are provided within a single .OCX file, the entire set of data types reported may not necessarily be applicable to the Control in question.

3.321 URL Property

Applies To: *HTMLRenderer*

Description

The URL property is a character vector that specifies the url of a resource to be requested by the *HTMLRenderer*. Its default value is an empty character vector.

Specifying URL is an alternative way to setting the *HTML* property in order to display content in the *HTMLRenderer*.

When you set the URL property, the *HTMLRenderer* will request the corresponding resource (from either the internet or the workspace via an *HTTPRequest* event) and the display will change according to the response. The *HTML* property is ignored and remains unchanged.

When you set the *HTML* property, the content of the *HTMLRenderer* will change accordingly. The current value of the URL property is ignored and remains unchanged.

If you set BOTH URL and *HTML* in the same statement, the value of URL takes precedence and the assignment to *HTML* is ignored (it remains unchanged).

3.322 Underline Property

Applies To: *Font*

Description

This property specifies whether or not the characters in the font associated with a *Font* object are underlined or not. It is either 0 (normal) or 1 (underlined). There is no default; the value of this property reflects the underline characteristic of the font allocated by Windows.

3.323 UndocksToRoot Property

Applies To: *CoolBand, Form, SubForm, ToolControl*

Description

Specifies the parent adopted by an object when its Type changes to a Form as a result of an undocking operation.

UndocksToRoot is a single number with the value 0 or 1.

If UndocksToRoot is 1, the object becomes a Form that is a child of Root and therefore becomes completely independent of the Form in which it was previously docked.

If UndocksToRoot is 0, the object becomes a Form that is a child of the Form in which it was previously docked and is therefore always displayed on top of it. This setting is appropriate for a dockable toolbar.

The default value of UndocksToRoot is 1 if the object was originally created as a child of Root; otherwise it is 0.

3.324 VAlign Property

Applies To: *Text*

Description

This property determines the vertical alignment of text in the *Text* object. It is either a single integer value, or, if the *Text* object has several components, a corresponding vector of such values.

These may be :

0	base aligned	the base line of the character is aligned on the y-coordinate specified by the <i>Points</i> property.
1	half aligned	the centre of the character is aligned on the y-coordinate specified by the <i>Points</i> property.
2	cap aligned	the top of the character is aligned on the y-coordinate specified by the <i>Points</i> property.
3	bottom aligned	the bottom of the character cell is aligned on the y-coordinate specified by the <i>Points</i> property.
4	top aligned	the top of the character cell is aligned on the y-coordinate specified by the <i>Points</i> property. This is the default.

3.325 VScroll Property

Applies To

Description

This property determines whether or not an object has a vertical scrollbar. It is a single integer with the value -2 , -1 , or 0 .

VScroll may only be set when the object is created with `WC` and may not subsequently be changed with `WS` or assignment.

For a *Form* object, the value -1 specifies that the *Form* has a vertical scrollbar. A value of 0 (which is the default) means that it does not.

When applied to an *Edit* object, the value -2 specifies that the data is scrollable vertically, but only by using the cursor keys; a scrollbar is not provided. A value of -1 causes a scrollbar to be displayed (whether or not one is needed).

When applied to a *List* object, the value -2 specifies that the data is scrollable vertically, but only by using the cursor keys; a scrollbar is not provided. A value of -1 causes a scrollbar to be displayed if required (when the list of items exceeds the height of the object).

When applied to a *Combo* or *ComboEx* object, a value of -1 or -2 causes a scrollbar to be displayed, whether or not one is required.

For all these objects, a value of 0 inhibits scrolling altogether.

For a *Scroll* object, *VScroll* may be -1 or 0 . If it is -1 the direction of the scrollbar is vertical. If both *HScroll* and *VScroll* are set to -1 , *HScroll* takes precedence and forces *VScroll* back to 0 .

For a *StatusBar*, *TabBar* or *ToolBar* with *Align* set to Left or Right, *VScroll* determines whether or not a vertical scrollbar is provided and how the object positions its children. If *VScroll* is 0 (the default) the object organises its children in multiple columns and does not provide a scrollbar. If *VScroll* is -1 or -2 , the object organises its children in a single column and provides a mini scrollbar to allow those positioned beyond the bottom edge of the object to be scrolled into view. If *VScroll* is -1 , the scrollbar is always shown. If *VScroll* is -2 , it is only shown when needed.

For a *Grid*, *VScroll* may be 0 (no vertical scrollbar), -1 (scrollbar is displayed when required), -2 (same as -1) or -3 (scrollbar is always displayed).

3.326 ValidIfEmpty Property

Applies To: *ButtonEdit*, *Edit*, *Spinner*

Description

This property applies to an *Edit* object with *Style* Single and specifies whether or not an empty field is considered to be valid. It also applies to a *Spinner*. Its value is either 0 (an empty field is not valid) or 1 (an empty field is valid). If the *FieldType* is Numeric, LongNumeric, Currency, Date or Time, the default value for *ValidIfEmpty* is 0 . Otherwise, its default value is 1 .

If *ValidIfEmpty* is 0 and the user attempts to *leave* the *Edit* object by shifting the input focus to another control, or by selecting a *Button* or *MenuItem*, the *Edit* object will generate a *BadValue* event. The *Text* property will reflect the appearance of the field and be empty, but the *Value* property will not be changed.

If *ValidIfEmpty* is 1 and the *FieldType* is Numeric, LongNumeric, Currency, date or Time, the *Value* property will be set to $\emptyset$ when the user clears the field and leaves it.

3.327 Value Property

Applies To: *ButtonEdit*, *Edit*, *Label*, *Spinner*

Description

This property specifies or reports the numeric value associated with an *Edit* or *Label* object whose *FieldType* property is set to *Numeric*, *LongNumeric*, *Date*, *LongDate* or *Time*.

If the *FieldType* is *Numeric* or *LongNumeric*, the *Value* property contains a scalar number. If the *FieldType* is *Date* or *LongDate*, the *Value* property is an integer representing the date as the number of days since 1st January 1900. If the *FieldType* is *Time*, the *Value* property is an integer that contains the number of seconds since midnight.

3.328 Values Property

Applies To: *Grid*

Description

This property specifies the data values for the cells in a *Grid* object. Values must be a matrix whose elements are either single numbers, character scalars, character vectors or character matrices. This property is updated as the user moves around the *Grid* changing data.

3.329 VariableHeight Property

Applies To: *CoolBar*

Description

The *VariableHeight* property specifies whether or not a *CoolBar* displays bands at the minimum required height, or all the same height (that of the largest).

VariableHeight is a single number with the value 0 (same height) or 1 (variable height). The default is 1.

3.330 View Property

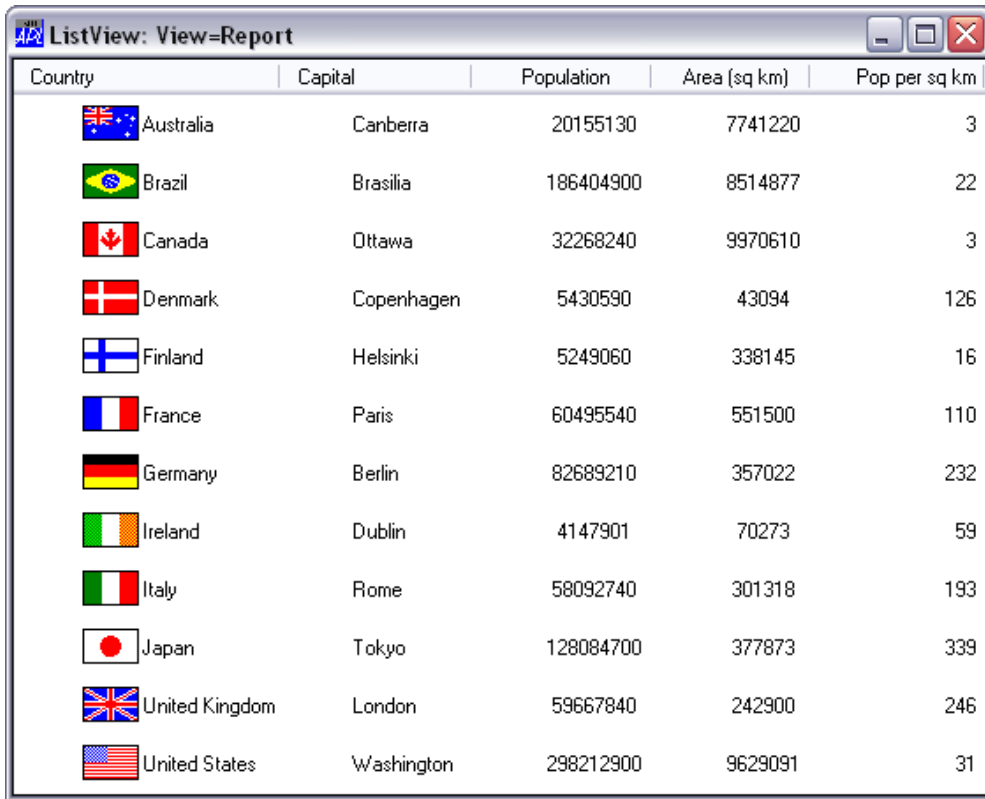
Applies To: *ListView*

Description

The *View* property specifies how the items in a *ListView* object are displayed. It is a character vector which may have one of the following values; `'Icon'` (the default), `'SmallIcon'`, `'List'` or `'Report'`.

When View is 'Icon' or 'SmallIcon', the items are arranged *row-wise* with large or small icons as appropriate. When View is set to 'List', the items are arranged *column-wise* using small icons. When View is set to 'Report', the items are displayed in a single column using small icons but with the matrix specified by *ReportInfo* displayed alongside. In this format, the ListView also provides column headings which are specified by the *ColTitles* property. The alignment of these titles (and of the data in the columns beneath them) is defined by the *ColTitleAlign* property. Examples of different views are illustrated below.





The screenshot shows a window titled "ListView: View=Report" with a standard APL interface (minimize, maximize, close buttons). The window contains a table with the following data:

Country	Capital	Population	Area (sq km)	Pop per sq km
 Australia	Canberra	20155130	7741220	3
 Brazil	Brasilia	186404900	8514877	22
 Canada	Ottawa	32268240	9970610	3
 Denmark	Copenhagen	5430590	43094	126
 Finland	Helsinki	5249060	338145	16
 France	Paris	60495540	551500	110
 Germany	Berlin	82689210	357022	232
 Ireland	Dublin	4147901	70273	59
 Italy	Rome	58092740	301318	193
 Japan	Tokyo	128084700	377873	339
 United Kingdom	London	59667840	242900	246
 United States	Washington	298212900	9629091	31

3.331 Visible Property

Applies To

Description

This property specifies whether or not an object is currently visible. It is a single number with the value 0 (object is invisible) or 1 (object is visible). The default is 1. Setting Visible on and off is a way to pop a dialog box up and down as required.

Note that an invisible object is not necessarily inactive, and is capable of generating events. For example, a *Button* with a *Cancel* property of 1 will generate a *Select* (30) event (if enabled) whether or not it is visible. An invisible object will also respond to methods and events sent to it by *Dyalog APL Language: Nq*.

3.332 WantsReturn Property

Applies To: *Edit*, *RichEdit*

Description

This Boolean property specifies the behaviour of the Enter key for a multi-line *Edit* (Style 'Multi') and a *RichEdit* object.

A value of 0 means that the Enter key is ignored by the *Edit* or *RichEdit*. Instead, the Enter key will (if appropriate) cause a Select event on a Button in the same Form. The user must press Ctrl+Enter to input a new line.

A value of 1 means that pressing the Enter key will introduce a new line into the object.

WantsReturn must be established when the object is created by `WC` and may not subsequently be altered using `WS`. Its default value is 0 in an *Edit* and 1 in a *RichEdit*.

3.333 WeekNumbers Property

Applies To: *Calendar*, *DateTimePicker*

Description

The WeekNumbers property specifies whether or not a *Calendar* object displays week numbers.

WeekNumbers is a single number with the value 0 (week numbers are *not* shown) or 1 (week numbers *are* shown); the default is 0.

3.334 Weight Property

Applies To: *Font*

Description

This property specifies the degree of boldness of a font associated with a *Font* object. It is a number in the range 0 to 1000, where 0 represents *very faint* and 1000 represents *very bold*. There is no default; the value of this property reflects the degree of boldness of the font allocated by Windows.

In general, 400 means *normal* and 700 means *bold*.

3.335 WordFormat Property

Applies To: *RichEdit*

Description

The *WordFormat* property is identical to the *CharFormat* property except that it is used to apply formatting to the selected word or words in a *RichEdit* object. If the selection is empty but the insertion point is inside a word, the formatting is applied to the word. See *CharFormat* for further details.

3.336 Wrap Property

Applies To: *ListView*, *ProgressBar*, *Spinner*, *UpDown*

Description

The *Wrap* property is Boolean and has a default value of 1.

For a *ListView* it specifies whether or not long labels (specified by the *Items* property) may be wrapped or not.

For a *ProgressBar* object it determines whether or not the object starts over again when it reaches its upper limit. In particular, if *Wrap* is 1, the value obtained when you set the *Thumb* property is given by the expression: `LIMITS[1]+THUMB|LIMITS[2]` where *THUMB* is the value to which you set the *Thumb* property and *LIMITS* is the value of the *Limits* property.

For a *Spinner*, *Wrap* determines what happens when the value in the *Spinner* reaches its upper or lower limit. If *Wrap* is 1 the *Spinner* will wrap around to its opposite limit. Otherwise it will stick.

3.337 XRange Property

Applies To

Description

XRange and *YRange* together determine a user-defined co-ordinate system. These properties are effective on the object's children which have *Coord* set to 'User'.

XRange is a 2-element numeric vector containing the x-coordinate of the top left and bottom right interior corners of the object respectively. See *Coord* for further details.

3.338 YRange Property

Applies To

Description

XRange and YRange together determine a user-defined co-ordinate system. These properties are effective on the object's children which have *Coord* set to 'User'.

YRange is a 2-element numeric vector containing the y-coordinate of the top left and bottom right interior corners of the object respectively. See *Coord* for further details.

3.339 Yield Property

Applies To: Root

Description

This property determines how frequently Dyalog checks the windows message queue. The message queue contains the underlying Windows messages for User interface events, such as mouse clicks, timers, and so on. Dyalog needs to check for the existence of these messages in order to pass them on to any GUI windows that may exist. In addition the windows message queue is used to handle interrupts and to implement the session window.

The value of Yield is an integer expressed in 1/1000's of a second. Its default value is 200. Yield defines the period of time allowed to elapse between the execution of successive lines of APL code before APL yields to Windows by requesting a message from the Windows queue. If Yield is set to zero APL does not explicitly yield.

Note that the value of this property only controls the yield frequency when APL is executing user-defined code. APL may also yield implicitly during *Dyalog APL Language: Dq*, *Dyalog APL Language: Nq*, *Dyalog APL Language: Wc*, `⌈SR`, *Dyalog APL Language: Ws* and *Dyalog APL Language: Wg* and in communicating with Auxiliary processors. Note that setting Yield to 0 (or to a very high value) during the execution of code that does not implicitly yield will effectively de-activate all other applications (including Program manager) and disable APL interrupts (Ctrl+Break). It should therefore be used with extreme caution.

4 Events

4.1 ActivateApp Event 139

Applies To: *Root*

Description

If enabled, this event is reported when the user switches to or from a Dyalog APL/W application.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ActivateApp' or 139
[3]	Activation flag	0 or 1

The Activation flag is 0 when the user switches *from* Dyalog APL to another application.

The Activation flag is 1 when the user switches *to* Dyalog APL from another application.

4.2 AddCol Event 153

Applies To: *Grid*

Description

If enabled, this event is reported by the *Grid* object if the user presses the Cursor Right key, and the current cell (*CurCell*) is within the last column on the *Grid*. The default action is to append a new column to the contents of the *Grid*. If you attach a callback function to this event and have it return a value of 0, a new column will not be appended to the *Grid*. Note that the event will not be generated unless the second element of the *AutoExpand* property is set to 1.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'AddCol ' or 153
[3]	Column number	number of the new column (integer)

An application may insert a new column into a *Grid* by calling `AddCol` as a method. The argument is a 1 to 7-element array as follows:

[1]	Column number	number of the new column (integer)
[2]	Column title	character vector or matrix
[3]	Column width	integer
[4]	Undo flag	0 or 1
[5]	Resize flag	0 or 1
[6]	Title colour	negative integer or 3-element RGB vector
[7]	Line type	integer

If you are using default column headings, *Column title* will be ignored and the columns will be re-labelled with the default titles. If you have set *ColTitles*, the title you specify will be inserted. If you omit *Column title*, a blank title will be inserted.

Similarly, if you have not previously set *CellWidths*, *ResizeCols*, *ColTitleFCol* or *ColLineTypes*, or if you have given them a scalar value, the corresponding parameter will be ignored. However, if you have specified *CellWidths*, *ResizeCols*, *ColTitleFCol* or *ColLineTypes* to be a *vector*, the number you specify in the corresponding parameter will be inserted into the appropriate property vector.

If you omit to specify *Column width* for the new column, it will be assigned a default value; new values for the other properties default to 0.

Undo flag (default 1) specifies whether or not the addition of the new column may subsequently be undone by an *Undo* event.

To insert a new column before the first one, you must specify the *Column number* as 1 (or 0 if `IO` is 0). To add a new column after the last one, you may specify any number greater than the current number of columns. The data in the new column will be set to 0 if the *Values* property is numeric, or to an empty character vector otherwise.

4.3 AddRow Event 152

Applies To: *Grid*

Description

If enabled, this event is reported by the *Grid* object if the user presses the Cursor Down key, and the current cell (*CurCell*) is within the last row on the *Grid*. The default action is to append a new row to the contents of the *Grid*. If you attach a callback function to this event and have it return a value of 0, a new row will not be appended to the *Grid*.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3 element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'AddRow' or 152
[3]	Row number	number of the new row (integer)

An application may insert a new row into a *Grid* by calling *AddRow* as a method. The argument is a 1 to 7-element array as follows:

[1]	Row number	integer
[2]	Row title	character vector or matrix
[3]	Row height	integer
[4]	Undo flag	0 or 1
[5]	Resize flag	0 or 1
[6]	Title colour	negative integer or 3-element RGB vector
[7]	Line type	integer

If you are using default row titles, *Row title* will be ignored and the rows will be re-labelled with default titles. If you *have* set *RowTitles*, the title you specify will be inserted. If you omit *Row title*, a blank title will be inserted.

Similarly, if you have not previously set *CellHeights*, *ResizeRows*, *RowTitleFCol* or *RowLineTypes*, or if you have given them a scalar value, the corresponding parameter will be ignored. However, if you have specified *CellHeights*, *ResizeRows*, *RowTitleFCol* or *RowLineTypes* to be a vector, the number you specify in the corresponding parameter

will be inserted into the appropriate property vector. If you omit *Row height*, it will be assigned a default value; new values for the other properties default to 0.

Undo flag (default 1) specifies whether or not the addition of the new row may subsequently be undone by an *Undo* event.

To insert a new row before the first one, you must specify the *Row number* as 1 (or 0 if $\square\text{IO}$ is 0). To add a new row after the last one, you may specify any number greater than the current number of rows. The data in the new row will be set to 0 if the *Values* property is numeric, or to an empty character vector otherwise.

4.4 AmbientChanged Event 533

Applies To: *ActiveXContainer*, *ActiveXControl*

Description

If enabled, this event is reported when any of the ambient properties change in an application hosting an *ActiveXControl* object. The new values of the ambient properties are available from the *FontObj*, *BCol* and *FCol* properties of the *ActiveXContainer*.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to -1 or by returning 0 from a callback function.

The event message reported as the result of $\square\text{DQ}$, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' AmbientChanged ' or 533
[3]	Property code	integer
[4]	Description	character vector

For properties supported by Dyalog APL, Property code and Description may be one of the following:

Property code	Description	Meaning
-701	DISPID_AMBIENT_BACKCOLOR	BCol has changed
-703	DISPID_AMBIENT_FORECOLOR	FCol has changed
-705	DISPID_AMBIENT_FONT	Font has changed
-1	DISPID_AMBIENT_UNKNOWN	Unknown

Note that other ambient properties may be reported, although these have no corresponding Dyalog APL property.

4.5 AnimStarted Event 294

Applies To: *Animation*

Description

If enabled, this event is reported by an *Animation* object just before an AVI clip starts playing

The event message reported as the result of `⌈DQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'AnimStarted' or 294

This event is reported for information only and cannot be disabled or modified in any way.

4.6 AnimStopped Event 295

Applies To: *Animation*

Description

If enabled, this event is reported by an *Animation* object just after an AVI clip has stopped playing

The event message reported as the result of `⌈DQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'AnimStopped' or 295

This event is reported for information only and cannot be disabled or modified in any way.

4.7 BadValue Event 180

Applies To: *ButtonEdit*, *Edit*, *Spinner*

Description

If enabled, this event is reported by an *Edit* or *Spinner* object when the user enters invalid data into the object and then switches focus to another control or to another application. Data is invalid if it conflicts with the *FieldType* property, or for a *Spinner* if it is outside the range specified by the *Limits* property.

The default action of the event is to sound the bell (beep). You can disable this action by returning 0 from a callback function or by setting its action code to `~1`. Note that in neither case is the *Value* property of the object updated. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'BadVaLue' or 180
[3]	Object name	character vector

The third element of the event message is either the name of the control to which the user has switched the focus, or is an empty vector if the focus has gone to another application.

4.8 BalloonHide Event 862

Applies To: *SysTrayItem*

Description

If enabled, this event is reported by an *SysTrayItem* object when a *BalloonTip* disappears for any reason other than it is dismissed by a timeout or because the user clicked the mouse.

The event message reported as the result of `OnEvent`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'BalloonHide' or 862

This event is reported for information only and cannot be disabled or modified in any way

4.9 BalloonShow Event 861

Applies To: *SysTrayItem*

Description

If enabled, this event is reported by an *SysTrayItem* object when a *BalloonTip* is displayed using the *ShowBalloonTip* method.

The event message reported as the result of `OnEvent`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'BalloonShow' or 861

This event is reported for information only and cannot be disabled or modified in any way

4.10 BalloonTimeout Event 863

Applies To: *SysTrayItem*

Description

If enabled, this event is reported by an *SysTrayItem* object when a *BalloonTip* is dismissed by a timeout or because the user clicked the *close* (X) button..

The event message reported as the result of `OnEvent`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'BalloonTimeout' or 863

This event is reported for information only and cannot be disabled or modified in any way

4.11 BalloonUserClick Event 864

Applies To: *SysTrayItem*

Description

If enabled, this event is reported by an *SysTrayItem* object when a *BalloonTip* is dismissed because the user clicked the mouse in the body of the *BalloonTip*. It is not reported when the user clicks the *close* (X) button.

The event message reported as the result of `DDQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'BalloonUserClick' or 864

This event is reported for information only and cannot be disabled or modified in any way

4.12 BeginEditLabel Event 300

Applies To: *ListView*, *TreeView*

Description

If enabled, this event is reported when the user clicks on an item in a *ListView* or *TreeView* object that has the focus, and signals the start of an edit operation. The default processing for the event is to display a pop-up edit box around the item and to permit the user to change its text.

You may disable the operation by setting the action code for the event to `-1`. You may prevent a particular item from being edited by returning 0 from a callback function. You may also initiate the edit operation by calling `BeginEditLabel` as a method.

The event message reported as the result of `WM_DBLCLK`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'BeginEditLabel' or 300
[3]	Item number	Integer. The index of the item.

4.13 CalendarDbClick Event 273

Applies To: *Calendar*

Description

If enabled, this event is reported when the user double-clicks the left mouse button over a *Calendar* object.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to -1 or by returning 0 from a callback function.

The event message reported as the result of `WM_DBLCLK`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CalendarDbClick' or 273
[3]	Item Number	integer
[4]	Mouse Button	integer
[5]	Shift State	integer. Sum of 1=shift key, 2=ctrl key, 4=Alt key
[6]	Element Type	integer

For the meaning of elements 3 and 6, see *CalendarDown*.

4.14 CalendarDown Event 271

Applies To: *Calendar*

Description

If enabled, this event is reported when the user depresses the left mouse button over a *Calendar* object.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to -1 or by returning 0 from a callback function.

The event message reported as the result of `CalendarDown`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CalendarDown' or 271
[3]	Item Number	integer (see below)
[4]	Mouse Button	integer
[5]	Shift State	integer. Sum of 1=shift key, 2=ctrl key, 4=Alt key
[6]	Element Type	integer (see below)

The 6th element of the event message is one of the following values:

If the value of the 6th element of the event message is 2 (CALENDARDATE), the 3rd element is the corresponding date reported as an *IDN*.

If the value of the 6th element of the event message is 5 (CALENDARDAY), the 3rd element is the index of the corresponding weekday (0-6).

If the value of the 6th element of the event message is 6 (CALENDARWEEKNUM), the 3rd element is the date of the first (leftmost) day in the corresponding week, reported as an *IDN*.

Otherwise, the 3rd element of the event message is 0.

4.15 CalendarMove Event 274

Applies To: *Calendar*

Description

If enabled, this event is reported when the user moves the left mouse button over a *Calendar* object.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to `-1` or by returning `0` from a callback function.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CalendarMove' or 274
[3]	Item Number	integer
[4]	Mouse Button	integer
[5]	Shift State	integer. Sum of 1=shift key, 2=ctrl key, 4=Alt key
[6]	Element Type	integer

For the meaning of elements 3 and 6, see *CalendarDown*.

4.16 CalendarUp Event 272

Applies To: *Calendar*

Description

If enabled, this event is reported when the user releases the left mouse button over a *Calendar* object.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to `-1` or by returning `0` from a callback function.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CalendarUp' or 272
[3]	Item Number	integer
[4]	Mouse Button	integer
[5]	Shift State	integer. Sum of 1=shift key, 2=ctrl key, 4=Alt key
[6]	Element Type	integer

For the meaning of elements 3 and 6, see *CalendarDown*.

4.17 CellChange Event 150

Applies To: *Grid*

Description

If enabled, this event is reported when the user changes the contents of a cell in a *Grid* object and then attempts to move to another cell or to another control outside the *Grid*.

The purpose of this event is to give the application the opportunity to perform additional validation before the update occurs (and to prevent it if necessary) or to update other cells in the *Grid* as a result of the change.

The default action for the CellChange event is to update the appropriate element of the *Values* property with the new data. This action can be disabled by returning 0 from the attached callback function. Notice however, that the user is not prevented from moving away from the cell. If you are using this event to perform additional validation and you require the user to correct the data before moving away, you may force the user back to the cell in question using the *CellMove* method.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is an 8-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'CellChange' or 150
[3]	Cell row	integer
[4]	Cell column	integer
[5]	New data	number or character array
[6]	Object name	character vector (name of object to which the user has transferred focus)
[7]	New cell (row)	integer
[8]	New cell (column)	integer

If the user moves to another cell in the *Grid*, the 6th element of the event message is the name of the *Grid* object and elements 7 and 8 specify the new cell address (␣IO dependent).

If the user switches the input focus to another control or selects a *MenuItem*, the 6th element of the event message contains the name of that control or *MenuItem*. If the user switches to another application, the 6th element of the event message is an empty character vector. In all these cases, the 7th and 8th elements are 0.

The 5th element of the event message contains the data value that will be used to update the *Values* property. This will be numeric if the *FieldType* of the associated *Edit* object is Numeric, LongNumeric, Date, LongDate or Time. Otherwise, it will be a character array.

An application can update an individual cell in the *Grid* under program control by calling *CellChange* as a method. If so, the *New object*, *New cell row* and *New cell column* parameters may be omitted.

4.18 CellChanged Event 164

Applies To: *Grid*

Description

If enabled, this event is reported after the user has changed the contents of a cell in a *Grid* object and then moved to another cell or to another control outside the *Grid*. The purpose of this event is to give the application the opportunity to perform calculations, and perhaps to update other cells in the *Grid* as a result of the change.

Note that this event is reported **after** the change has taken place, and after the *Values* property has been updated. Furthermore, neither setting the event action code to `-1` nor returning 0 from a callback function has any effect. If you wish to *validate* the new data you should use the *CellChange* (150) event instead.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is an 5-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'CellChanged' or 164
[3]	Cell row	integer
[4]	Cell column	integer
[5]	New data	number or character array

The 5th element of the event message contains the data value that has been used to update the *Values* property. This will be numeric if the *FieldType* of the associated Edit object is Numeric, LongNumeric, Date, LongDate or Time. Otherwise, it will be a character array.

If you want to update an individual cell under program control, you may call *CellChange*, but not *CellChanged*, as a method.

4.19 CellDbClick Event 163

Applies To: *Grid*

Description

If enabled, this event is reported when the user double-clicks a mouse button whilst over a cell in a *Grid*. The purpose of this event is to allow an application to enable some special action on double-click. This event may not be disabled.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 9 element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CellDbClick' or 163
[3]	Y	y-position of mouse (number)
[4]	X	x-position of mouse (number)
[5]	Button	button pressed (number) 1 = left button 2 = right button 4 = middle button
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down 4 = Alt key is down
[7]	Cell row	integer
[8]	Cell column	integer
[9]	Title index	integer

The y and x position of the mouse are reported relative to the top-left corner of the *Grid*.

The cell row and column are $\square$ IO dependent

If the user clicks over a row *title*, the value reported for the column is -1 , and the value reported for Title index is the index of that row title in *RowTitles*, or, if *RowTitles* is not defined, the row number. Column titles are handled in a similar fashion.

4.20 CellDown Event 161

Applies To: *Grid*

Description

If enabled, this event is reported when the user presses a mouse button down whilst over a cell in a *Grid*. The purpose of this event is to allow an application to display a pop-up *Menu* or a *Locator* over a cell in a *Grid* or to take some other special action.

The default action is to generate a *CellMove* event which will then position the user on the new cell. This action can be prevented by returning 0 from the callback function, in which case the normally ensuing *CellMove* event will not occur.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is an 9 element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CellDown' or 161
[3]	Y	y-position of mouse (number)
[4]	X	x-position of mouse (number)
[5]	Button	button pressed (number) 1 = left button 2 = right button 4 = middle button
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down 4 = Alt key is down
[7]	Cell row	integer
[8]	Cell column	integer
[9]	Title index	integer

The y and x position of the mouse are reported relative to the top-left corner of the *Grid*.

The cell row and column are $\square$ IO dependent

If the user clicks over a row *title*, the value reported for the column is -1 , and the value reported for Title index is the index of that row title in *RowTitles*, or, if *RowTitles* is not defined, the row number. Column titles are handled in a similar fashion.

An application **can** position the user on a particular cell in a *Grid* by calling *CellDown* as a method, but it is recommended that a *CellMove* event is used instead.

4.21 CellError Event 157

Applies To: *Grid*

Description

If enabled, this event is reported when the user inserts invalid data into the *Edit* object associated with a cell in a *Grid* object and then attempts to move to another cell or to another control outside the *Grid*. It is also reported if the user selects a *MenuItem*.

The default action for the `CellError` event is to sound the bell (beep). This action can be disabled by returning 0 from the attached callback function. Whatever the result of the callback, the user will be prevented from moving to another cell in the *Grid* and the *CurCell* and *Values* properties will remain unchanged. The user is not prevented from switching to any other control or to another application. However, if and when the user returns to the *Grid*, the current cell (*CurCell*) remains the invalid one and the user may not select a different one until the invalid data in the cell has been corrected. If you wish to allow the user to move to another cell without correcting the data, you may do so by generating a *CellMove* event explicitly. However, the *Values* property will remain unchanged and the invalid contents of the *Edit* object will simply be discarded.

The event message reported as the result of *Dyalog APL Language: Dq* or supplied as the right argument to your callback function, is an 8-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'CellError' or 157
[3]	Cell row	integer
[4]	Cell column	integer
[5]	Invalid data	character vector
[6]	Object name	character vector (name of object to which the user has transferred focus)
[7]	New cell (row)	integer
[8]	New cell (column)	integer

If the user moves to another cell in the *Grid*, the 6th element of the event message is the name of the *Grid* object and elements 7 and 8 specify the cell address (□IO dependent).

If the user switches the input focus to another control or selects a *MenuItem*, the 6th element of the event message contains the name of that control or *MenuItem*. If the user switches to another application, the 6th element of the event message is an empty character vector. In all these cases, the 7th and 8th elements are 0.

The 5th element of the event message contains the character vector in the *Text* property of the associated *Edit* object which is inconsistent with its *FieldType*.

4.22 CellMove Event 151

Applies To: *Grid*

Description

If enabled, this event is reported when the user attempts to position the cursor over a cell in a *Grid* by clicking the left mouse button or by pressing a cursor movement key. The purpose of this event is to allow an application to perform some action prior to the user entering a cell, to inhibit entry to a cell, or to deny exit from the current cell.

The default action is to position the user on the new cell. This action can be prevented by returning a 0 from the callback function attached to the event.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is an 8 element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CellMove' or 151
[3]	New cell row	integer
[4]	New cell column	integer
[5]	Scroll flag	0 or 1
[6]	Selection flag	0, 1 or 2
[7]	Mouse flag	0 or 1
[8]	Changed flag	0 or 1 (relates to current cell)
[9]	New value	new value of current cell or 0

The 5th element of the event message is 1 if switching to the new cell would cause the *Grid* to scroll.

The 6th element of the event message is 1 if the user is moving to the new cell by extending the selection. It is 2 if the user selects an entire row or column (by clicking on a title), which moves the current cell to the first one in the selection.

The 7th element of the event message is 1 if the mouse is used to switch to a new cell.

The 8th element of the event message is 1 if the user is attempting to move to the new cell from another cell in the *Grid* having typed in (as if to alter) the current cell.

The 9th element of the event message is the intended new value in the current cell or $\emptyset$ (zilde) if *Changed flag* is 0.

The *CellMove* event may be used to validate and refuse changes as the user navigates between cells. See also *CellChange*.

An application can position the user on a particular cell in a *Grid* by calling *CellMove* as a method. If so, the argument need contain only the *New cell row* and *New cell column* parameters.

4.23 CellOver Event 160

Applies To: *Grid*

Description

If enabled, this event is reported when the user moves the mouse pointer whilst over a cell in a *Grid*.

There is no default action for this event.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is an 9 element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CellOver' or 160
[3]	Y	y-position of mouse (number)
[4]	X	x-position of mouse (number)
[5]	Button	button pressed (number) 1 = left button 2 = right button 4 = middle button
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down 4 = Alt key is down
[7]	Cell row	integer
[8]	Cell column	integer
[9]	Title index	integer

The y and x position of the mouse are reported relative to the top-left corner of the *Grid*.

The cell row and column are `IO` dependent

If the user moves the mouse pointer over a row *title*, the value reported for the column is `-1`, and the value reported for Title index is the index of that row title in *RowTitles*, or, if *RowTitles* is not defined, the row number. Column titles are handled in a similar fashion.

4.24 CellUp Event 162

Applies To: *Grid*

Description

If enabled, this event is reported when the user **releases** a mouse button down whilst over a cell in a *Grid*. This event is a companion to the *CellDown* event and could be used to hide a pop-up which was displayed in response to the *CellDown*. The *CellUp* event performs no default action and may not be disabled.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 9 element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CellUp' or 162
[3]	Y	y-position of mouse (number)
[4]	X	x-position of mouse (number)
[5]	Button	button released (number) 1 = left button 2 = right button 4 = middle button
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down 4 = Alt key is down
[7]	Cell row	integer
[8]	Cell column	integer
[9]	Title index	integer

The y and x position of the mouse are reported relative to the top-left corner of the *Grid*.

The cell row and column are $\square$ IO dependent.

If the user clicks over a row *title*, the value reported for the column is -1 , and the value reported for Title index is the index of that row title in *RowTitles*, or, if *RowTitles* is not defined, the row number. Column titles are handled in a similar fashion.

An application **can** position the user on a particular cell in a *Grid* by calling *CellDown* event as a method, but it is recommended that a *CellMove* event is used instead.

4.25 Change Event 36

Applies To: *ButtonEdit, Combo, Edit, RichEdit, Spinner*

Description

If enabled, this event is reported when the user alters the text in a *Combo* or *Edit* object (by typing). The event is not applicable for a *Combo* with *Style 'Drop'* because this *Style* does not allow the user to alter data. The *Change* event is not reported repeatedly as the user edits the data. Instead, it is reported when the user indicates that he has finished with the field by :

1. clicking on another object, **or**
2. causing an event on another object (without altering the input focus) which will fire a callback function or cause *Dyalog APL Language: Dq* to terminate. This can occur if the user chooses a *MenuItem*, or fires a *Button* with the *Default* or *Cancel* property by pressing Enter or Esc, or selects an object using an accelerator key.

The purpose of the *Change* event is to allow the application to validate data which has been newly entered to the field, before proceeding with another action. It is for this reason that the event is fired not just when the input focus changes, but also when the user takes some action that could cause the application to do something else.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Change' or 36
[3]	Object name	character vector (name of object that is to receive the focus or generate an event)

If the focus is transferred to an external application, the third element is an empty vector.

The default processing for this event is to allow the focus to change (if applicable) and to reset the internal flag that indicates that the data in the field has changed.

If you disable the event by setting the "action" code to `-1`, or inhibit it by returning 0 from a callback, the focus change (if applicable) is allowed to proceed, but the internal flag is not reset. If you wish to prevent the focus change you must explicitly reset the focus back onto the object that generated the event.

If the event was generated because the user switched to another application, and you return a 0 from your callback (because the data was not valid), the flag marking the *Combo* or *Edit* as having been changed remains set. If the user returns to your application by re-focusing on the same *Combo* or *Edit*, nothing happens immediately, but because the field is marked as changed (the flag was not reset) you will get another

Change event when he leaves it. However, if the user returns to your application in some other way, for example, by focusing on another object or by selecting a *MenuItem*, a second Change event will be generated immediately.

The following function illustrates how Change events can be processed. The *Check* function referred to in line[4] is assumed to return 1 if the data is valid and 0 if not.

```
[0]  R←VALIDATE Msg
[1]  A Validates field contents after Change event
[2]
[3]  A Normal exit (R←1) if data is valid
[4]  →(R←Check>Msg)/Exit
[5]
[6]  A R now 0, so field remains marked as "changed"
[7]
[8]  A If user has switched to another application,
[9]  A we need take no further action because we will
[10] A get a second Change event when he returns.
[11] →(''≡3>Msg)/Exit
[12]
[13] A Display error box (prepared earlier)
[14] 'ERR' □WS 'Text' 'Data is invalid'
[15] □DQ'ERR'
[16]
[17] A Restore focus to bad field
[18] □NQ(>Msg)40
[19]
[20] Exit:
```

4.26 ClickComment Method 225

Applies To: *Grid*

Description

If enabled, a ClickComment event is generated when the user clicks the mouse in a comment window of a *Grid*.

The event message reported as the result of □DQ, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ClickComment' or 225
[3]	Row	integer
[4]	Column	integer

The event message reports the co-ordinates of the cell. The default action is to raise the comment window so that it appears above all other, potentially overlapping, comment windows.

Note that if the comment window relates to a row or column *title*, the value reported in element [3] or [4] of the event message is `-1`.

4.27 ClipChange Event 120

Applies To: *Clipboard*

Description

If enabled, this event is reported when another application changes the contents of the Windows clipboard.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ClipChange' or 120

4.28 Close Event 33

Applies To

Description

A Close event is generated when an object is destroyed.

For a *Form* or *SubForm*, the event may be generated by the user selecting "Close" from its System Menu. In this case, the event is reported **before** the window is destroyed, and you may prevent it from going ahead by associating a callback function which returns a result of 0.

By trapping this event you can control termination of your application in many different ways. For example, you could:

- automatically close all *Forms* in your application when the master *Form* is closed.
- prevent the user from terminating the application if it is inappropriate at that time.
- display an "Are you sure ?" *MsgBox*.

Note that although all the object's child GUI objects will also be destroyed, none of the child objects will generate a Close event.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Close' or 33

4.29 CloseUp Event 46

Applies To: *DateTimePicker*

Description

If enabled, this event is reported by a *DateTimePicker* object just before the drop-down calendar is hidden. It applies only if the Style of the *DateTimePicker* is 'Combo'.

The event message reported as the result of □DQ, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CloseUp' or 46

This event is reported for information only and cannot be disabled or modified in any way.

4.30 ColorChange Event 430

Applies To: *ColorButton*

Description

If enabled, this event is reported by a *ColorButton* object when the user chooses a colour from the colour selection drop-down.

The event message reported as the result of `□□□`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ColorChange' or 430
[3]	New Colour	3-element integer vector

The 3rd element of the event message contains the RGB value for the selected colour.

Note that the event is reported when the user chooses a colour, whether or not the newly selected colour differs from the one that was previously selected.

This event is reported for information only and cannot be disabled or modified in any way.

4.31 ColumnClick Event 320

Applies To: *ListView*

Description

If enabled, this event is reported when the user clicks on the column heading in a *ListView* object. This event may not be disabled or affected by a callback function in any way.

The event message reported as the result of `□□□`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ColumnClick' or 320
[3]	Column number	Integer
[4]	Button	button pressed (number) 1 = left button 2 = right button 4 = middle button
[5]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down

4.32 Configure Event 31

Applies To

Description

If enabled, this event is generated when the configuration of an object (position and/or size) is about to change.

For a *Form*, the event is generated when the *Form* is resized or moved by the user.

For any object other than a *Form*, it can occur in one of two ways. Firstly, whenever a *Form* is resized, the system (by default) re-arranges its children so as to maintain their relative position and size. This generates a Configure event (if enabled) for each one of them.

Secondly, it can occur as a result of the user resizing the object directly. This facility is enabled by setting the object's *Sizeable* property to 1.

Note that a Configure event is **not** reported when an object is moved using "drag & drop". See *Dragable* (property) and *DragDrop* (event) for details of this operation.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Configure' or 31
[3]	Y	y-position of top left corner
[4]	X	x-position of top left corner
[5]	H	height of object
[6]	W	width of object

For any object, the operation can be prevented by returning a scalar 0 from the callback function associated with the Configure event.

Full-Drag Considerations

The user may choose a system option, described here as *full-drag*, whereby the contents of the window are re-arranged during a resize operation.

If you manage the geometry of your controls using the Attach property, APL honours *full drag* during resize, changing the size and position of your controls dynamically for you.

However, if you manage the geometry of controls using Configure event callbacks, you should consider the following.

1. If full drag is in enabled, APL generates Configure events *during* the resize operation, allowing you to dynamically alter the geometry of controls as you wish. However, the following restrictions apply:
2. Configure callbacks will only be executed when the interpreter is *idle*. For example during a `⎕DQ` or during Session input (6-space prompt). If the user attempts to move/resize a window that has Configure callbacks attached when the interpreter is busy, the move/resize is not started. This is similar to the operation in non *full drag* mode, where the move/resize is allowed but the callback does not execute until the interpreter again becomes idle.
3. The callback cannot be traced. It is necessary to debug the callback code with *full drag* disabled.
4. Any untrapped errors in the Configure callback will not halt execution in the normal way, but will instead be reported in the *Status Window*. Note that it is also not possible to trap such errors higher up the SI stack than the Configure Callback.
5. There are some programming styles to be avoided if *full drag* Configure callbacks are to be processed correctly. For example events generated by monadic `⎕NQ`

within a Configure callback will not be processed until the entire resize operation has been completed.

6. It is not possible to save a workspace from within a Configure Callback in *full drag* mode.

The above restrictions apply to Configure events when *full drag* is enabled, but *only* when *full drag* is enabled. The behaviour of Configure callbacks with *full drag* disabled is the same as for other events.

4.33 ContextMenu Event 410

Applies To

Description

If enabled, this event is reported when the user performs the standard Windows action to display a ContextMenu. These include clicking/releasing the right mouse button and pressing F10.

If the object has its own standard context menu, for example an Edit object, the default action is to display this menu. If the object is dockable (see *Docking Property*), the default action is to display the standard (English) Dyalog APL docking menu.

You may use this event to display your own pop-up context menu, by `⌈⌋`ing it within a callback function. In this case, your callback function should return 0 to disable the standard context menu.

The event message reported as the result of `⌈⌋`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ContextMenu' or 410
[3]	(reserved)	Empty character vector
[4]	Y	y-position of the mouse (number)
[5]	X	x-position of the mouse (number)

4.34 Create Event 34

Applies To

Description

If enabled, this event is reported *after* an object has been created. You may not nullify or modify the event with a 0-returning callback, nor may you generate the event using `⎕NQ`, or call it as a method.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Create' or 34
[3]	Flag	1 = object was created by <code>⎕WC</code> 0 = object was created by <code>)LOAD</code> , <code>)COPY</code> or <code>⎕OR</code>

This event also applies to the Session object `⎕SE` and may be used to fire a start-up function (in the `⎕SE` namespace) when APL initialises.

4.35 DDE Event 50

Applies To: *Root*

Description

If enabled, a DDE event is generated whenever a DDE message is received by Dyalog APL. This will occur whenever a server notifies APL that the value of a shared variable has changed, and whenever a client application requests data from APL. If you have several shared variables, you can determine which of them has changed or whose value has been requested using `⎕SVS`.

This event **only** applies to the *Root* object `"."`, so to enable it you must execute one of the following statements :

```

'.' ⎕WS 'Event' 50 1
'.' ⎕WS 'Event' 50 fn
'.' ⎕WS 'Event' 50 fn larg

```

The first statement would cause *Dyalog APL Language: Dq* to terminate on receipt of a DDE event. The second would cause it to call `"fn"` each time. The third would do likewise but the value in `"larg"` would be supplied as its left argument.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DDE ' or 50

Note

Due to the nature of DDE "conversations" messages may be received when in fact no change in the value of any shared variables has occurred. Your application code must therefore be prepared to cater for this situation.

4.36 DateTimeChange Event 267

Applies To: *DateTimePicker*

Description

If enabled, this event is reported by a *DateTimePicker* object when the user changes the *DateTime* value. This occurs when the user selects a new date from the drop-down calendar, or increments or decrements a date time element using the spinner buttons, or edits a datetime element using the keyboard. In the latter case, the event may not be generated until the input focus leaves the corresponding date time element.

The event message reported as the result of `GetDDE`, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DateTimeChange ' or 267
[3]	IDN	integer
[4]	Hour	integer
[5]	Minute	integer
[6]	Second	integer

This event is reported for information only and cannot be disabled or modified in any way. It is not possible to enqueue this event; set the *DateTime* value instead.

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

4.37 DisplayChange Event 137

Applies To: *Root*

Description

If enabled, this event is reported when the user changes the screen resolution or number of colours. The event is reported for information only; you cannot prevent the change from occurring.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DisplayChange' or 137
[3]	Height	Integer. Number of pixels in the y-direction
[4]	Width	Integer. Number of pixels in the x-direction
[5]	Number of colours	Integer.

4.38 DoPopup Event 846

Applies To: *HTMLRenderer*

Description

This event is triggered when the *HTMLRenderer* client attempts to open a new window. This could be fired by an HTML `<a>` tag with the target attribute set to open a URL in a new window or by a JavaScript `window.open()` call. Note that this does not apply to JavaScript Popup Boxes.

By default the *HTMLRenderer* ignores a request for a new window, but if the *DoPoup* event, is enabled, it provides the information needed to process the request in the workspace.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'DoPopup' or 846
[3]	URL	the requested url
[4]	Attributes	requested window attributes (see below)
[5]	Framename	character vector framename

Attributes is a 7-element nested vector that specifies the requested attributes for the new window. The HTMLRenderer currently provides no mechanism to use this information.

[1]	2-element vector of top, left positions – positions not specified are 0
[2]	2-element vector of height, width – sizes not specified are 0
[3]	Integer "WindowDisposition". See https://magpcss.org/ceforum/apidocs3/projects/(default)/cef_window_open_disposition_t.html .
[4]	Boolean menubar (default=1)
[5]	Boolean scrollbar (default=1)
[6]	Boolean statusbar (default=1)
[7]	Boolean location/toolbar (default=1)

To respond to the request for a new window, the callback function should open the requested URL as appropriate, for example, in a newly created *HTMLRenderer* object.

EXAMPLE

```
'h'[]WC 'HTMLRenderer'
'h'[]WS ('Event' 'DoPopup' 'DoPopupCB')

html← '

```

```
▼ DoPopupCB msg;props;posn
[1] props←<'URL'(3>msg)
[2] (posn size)←2↑4>msg
[3] props,←('Posn'posn)('Size'size)
[4] 'popup'[]WC'HTMLRenderer'props
▼
```

EXTENDED EXAMPLE

```

    ▽ {r}←DoPopupDemo args;html;h;c;s;e;p;d
[1]   →EndHTML
[2]   StartHTML:
[3]   A<html>
[4]   A  <head><title>DoPopupDemo</title></head>
[5]   A<body>
[6]   A  {}
[7]   A  <button type="button"
onclick="window.open('callback','_blank','width=400,height=200')">Click
me!</button>
[8]   A</body>
[9]   A</html>
[10]  EndHTML:
[11]  html←ε'A'(1+StartHTML)↓EndHTML↑NR=SI
[12]  :If 0∈args
[13]      renderers←0
[14]      html←'{'R'This is the original window'+html
[15]      s←'Size'(0 0)
[16]      p←'Posn'(0 0)
[17]      d←'Data' 0
[18]      →Create
[19]  :Else
[20]      :Select 2>args
[21]      :Case 'DoPopup'
[22]          d←'Data'(1+[/renderers.Data)
[23]          html←'{'R('This is window ',2>d)+html
[24]          s←'Size'(2>4>args)
[25]          p←'Posn'(100+(>args).Posn)
[26]          →Create
[27]      :Case 'Close'
[28]          renderers~←>args
[29]      :EndSelect
[30]  :EndIf
[31]  →0
[32]  Create:
[33]  h←'HTML'html
[34]  c←'Coord' 'Pixel'
[35]  e←'Event'('onAll' 'DoPopupDemo')
[36]  renderers,←NEW'HTMLRenderer'(h c s p e d)
    ▽

```

The example function show above will display a new window when the button labelled Click me! is pressed. To start, type:

```
DoPopupDemo ' '
```

4.39 DockAccept Event 483

Applies To: *CoolBand, CoolBar, Form, SubForm, ToolControl*

Description

If enabled, this event is reported by a host object just before it accepts a client object docking operation. This event is reported (by the host) immediately after the *DockRequest* is reported (by the client).

The event message reported as the result of `□□□□□□□□`, or supplied as the right argument to your callback function, is a 7-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DockAccept ' or 483
[3]	Client Object	ref or character vector
[4]	Edge	character vector
[5]	y-position	number
[6]	x-position	number
[7]	Outline rectangle	4-element nested

Elements 4-7 of this event message are the same as those reported by *DockMove*, and the effect of a callback function is identical. See *DockMove* for further information.

4.40 DockCancel Event 485

Applies To: *CoolBand, CoolBar, Form, SubForm, ToolControl*

Description

If enabled, this event is reported by a client object when the user aborts a docking operation by pressing Escape.

The event message reported as the result of `OnDock`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DockCancel ' or 485

This event is reported for information only and cannot be cancelled or inhibited in any way.

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

4.41 DockEnd Event 484

Applies To: *CoolBand, CoolBar, Form, SubForm, ToolControl*

Description

If enabled, this event is reported by a client object after it has been successfully docked in a host object.

The event message reported as the result of `OnDock`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DockEnd ' or 484

This event is reported for information only and cannot be cancelled or inhibited in any way.

4.42 DockMove Event 481

Applies To: *CoolBand, CoolBar, Form, SubForm, ToolControl*

Description

If enabled, this event is reported by a host object when a dockable object (the client) is dragged over it. The event will only be reported if the name of the client object is included in the list of objects that the host object will accept, which is defined by its `DockChildren` property.

The event message reported as the result of `OnDock`, or supplied as the right argument to your callback function, is a 7-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DockMove' or 481
[3]	Client Object	ref or character vector
[4]	Edge	character vector
[5]	y-position	number
[6]	x-position	number
[7]	Outline rectangle	(see below)

The 4th element of the event message `Edge` is a character vector that indicates along which edge of the host object the client object will be docked if the mouse button is released. It is either 'Top', 'Bottom', 'Left', 'Right' or 'None'. The latter indicates that the object will not be docked. An object will dock only if the mouse pointer is inside, and sufficiently near to an edge of, the host.

The 5th and 6th elements of the event message report the position of the mouse pointer in the host object.

The 7th element of the event message is a 4-element nested vector containing the y-position, x-position, height and width of a rectangle. If `Edge` is 'None', this is the bounding rectangle of the client object. Otherwise, the rectangle describes a docking zone in the host that the client object will occupy when the mouse button is released.

If a callback function returns 0, the system displays the bounding rectangle and not a docking zone, and the docking operation is inhibited. You could use this mechanism to prohibit docking along one or more edges, whilst allowing it along others.

A callback function may modify the event message to cause a different sized docking zone to be displayed, or to force docking along a particular edge.

The `DockMove` event is generated repeatedly as the docking object is dragged.

4.43 DockRequest Event 482

Applies To: *CoolBand, CoolBar, Form, SubForm, ToolControl*

Description

If enabled, this event is reported by a client object just before it is docked in a host object, when the user releases the mouse button.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 7-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DockRequest' or 482
[3]	Host Object	ref or character vector
[4]	Edge	character vector
[5]	y-position	number
[6]	x-position	number
[7]	Outline rectangle	4-element nested

Elements 4-7 of this event message are the same as those reported by *DockMove*, and the effect of a callback function is identical. See *DockMove* for further information.

4.44 DockStart Event 480

Applies To: *CoolBand, CoolBar, Form, SubForm, ToolControl*

Description

If enabled, this event is reported by a dockable object (one whose *Dockable* property is set to 1) when the user starts to drag it using the mouse.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DockStart' or 480

A callback function may prevent the docking operation from starting by returning 0.

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

4.45 DragDrop Event 11

Applies To

Description

If enabled, this event is reported when the user drops one object (the *dragged object*) onto another (the *target object*). The event is generated by the target object, that is, the object which is being dropped **on**.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 7-element vector as follows :

[1]	Target Object	ref or character vector
[2]	Event	'DragDrop' or 11
[3]	Dragged Object	ref or character vector
[4]	Y	y-position of mouse pointer in the target object
[5]	X	x-position of mouse pointer in the target object
[6]	H	height of dragged object in the coordinate system of the target object
[7]	W	width of dragged object in the coordinate system of the target object
[8]	Shift State	numeric scalar containing the sum of the values associated with the Shift(1), Ctrl(2) and Alt(4) keys when the object was dropped.

Note that Y, H, X and W are all reported relative to the target object.

4.46 DropDown Event 45

Applies To

Description

If enabled, this event is reported when the user clicks the drop-down button in a *Combo*, *ComboEx*, *ColorButton*, *DateTimePicker* or *Menu* object, just before the drop-down list, colour selection, calendar or -menu is displayed.

For a *Button* this event only applies if the *Style* of the *Button* is *Split*. For such a *Button* and for the *ButtonEdit* object there is no default processing for the event and it is the responsibility of the programmer to take appropriate action in a call-back function.

For a *DateTimePicker* this event only applies if the *Style* of the *DateTimePicker* is *'Combo'*.

The event message reported as the result of `WM_DROPDOWN`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DropDown' or 45

This event is reported for information only and cannot be disabled or modified in any way.

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

4.47 DropFiles Event 450

Applies To

Description

If enabled, this event is reported when the user drags a file icon or a set of file icons and drops them onto the object. The system takes no action other than to report the event.

Note that the event is only reported if *AcceptFiles* is set to 1.

The event message reported as the result of `WM_DROPFILES`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DropFiles' or 450
[3]	Files	Vector of character vectors containing the file names.
[4]	Item number	Integer. The index of the item within the object onto which the file(s) was dropped. Applies only to objects that have an <i>Items</i> property such as List, <i>ListView</i> and <i>TreeView</i> .
[5]	Shift state	Integer. Sum of 1=shift key, 2=Ctrl key, 4=Alt key

4.48 DropObjects Event 455

Applies To

Description

If enabled, this event is reported when the user drags an object icon or a set of object icons from the *Explorer Tool* or *Find Objects Tool* (which are part of the Dyalog APL Session) and drops them onto the object. The system takes no action other than to report the event. You can use this event to extend drag-drop functionality in your Session. For example, you could perform an operation by drag-dropping an APL object icon onto a Button in the Session toolbar.

The event message reported as the result of □DQ, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DropObjects' or 455
[3]	Objects	Vector of character vectors containing the object names.
[4]	Item number	Integer. The index of the item within the object onto which the file(s) was dropped. Applies only to objects that have an <i>Items</i> property such as List, <i>ListView</i> and <i>TreeView</i> . Otherwise this value is -1.
[5]	Shift state	Integer. Sum of 1=Shift key, 2=Ctrl key, 4=Alt key

4.49 DyalogCustomMessage1 Event 95

Applies To: *Form*

Description

This event allows external applications and dynamic link libraries to insert events into the Dyalog APL/W message queue.

DyalogCustomMessage1 may be invoked from a C program as follows:

```
msg=RegisterWindowMessage( "DyalogCustomMessage1" );
SendMessage(hWnd,msg,wParam,lParam);
```

where hWnd is the window handle of the object in the Dyalog APL Workspace. If the object is a *Form*, this may be obtained using FindWindow(). If not, hWnd may be passed to the external process as an argument to a function.

The parameters wParam and lParam are reported as numeric arguments to the APL callback function.

NOTE: It is not possible to pass pointers to data in wParam or lParam. When the APL callback executes the pointers may not be valid.

If a callback function is attached to the event, the callback function will be run when the event reaches the top of the queue.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to -1 or by returning 0 from a callback function.

The result of a callback function is **not** returned to the external application.

The event message reported as the result of □DQ, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DyalogCustomMessage1' or 95
[3]	wparam	integer
[4]	lparam	integer

4.50 EndEditLabel Event 301

Applies To: *ListView*, *TreeView*

Description

If enabled, this event is reported when the user signals completion of an edit operation in a *ListView* or *TreeView* object. This occurs when the item being edited loses the focus or when the user presses the Enter key. The default processing for the event is to update the item label (string) with the edited text in the pop-up edit box.

You may disable the update operation by setting the action code for the event to -1. You may also prevent the update from occurring by returning 0 from a callback function. You may specify the text used to update the item by returning the event message (containing the desired text) from a callback function. Finally, you may change the text of any item dynamically by calling `EndEditLabel` as a method.

The event message reported as the result of `OnDQ`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'EndEditLabel' or 301
[3]	Item number	Integer. The index of the item.
[4]	Text	character vector containing the text that will be used to update the item's label.

4.51 EndSplit Event 282

Applies To: *Splitter*

Description

If enabled, this event is reported when the user releases the left mouse button to signify the end of a drag operation on a *Splitter* object.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to -1 or by returning 0 from a callback function.

The event message reported as the result of `OnDQ`, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'EndSplit' or 282
[3]	Y	y-position of top left corner
[4]	X	x-position of top left corner
[5]	H	height of the Splitter
[6]	W	width of the Splitter

See also *StartSplit*, *Splitting*.

4.52 ExitApp Event 132

Applies To: *Root*

Description

If enabled, this event is reported when the user attempts to terminate a Dyalog APL/W application from the Windows Task List.

The Windows Task list displays the names of all running applications. The name displayed for a Dyalog APL/W application is defined by the *Caption* property of the system object *Root*. If you fail to define this property, there will be no entry for the application in the Task List.

If you wish to prevent the user from terminating your application from the Windows Task List, you may disable this event by setting its action code to `-1`. However, if you do this, your user may be puzzled as to why the operation does not work as expected. An alternative is to attach a callback function to the event which displays a message box. Not only does this allow you to provide user feedback, but you can provide confirm/cancel options. If your callback function returns a zero, your application will not be terminated.

Note that this event only provides for termination via the Windows Task List. See also the *ExitWindows* event.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ExitApp' or 132

4.53 ExitWindows Event 131

Applies To: *Root*

Description

If enabled, this event is reported when the user attempts to terminate the Windows Operating System. When this is done, Windows gives all running applications the opportunity to prevent it. Typically, an application that has unsaved changes will display a dialog box warning the user of this situation and offering the opportunity to cancel the termination. The default action for this event is to allow Windows to close. You can prevent this by returning a zero from a callback function. You can also prevent the user from closing Windows down by disabling the event altogether. This is achieved by setting its action code to `-1`. In most cases this is less preferable than the callback method as it does not allow you to inform the user as to why Windows won't terminate.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ExitWindows' or 131
[3]	Flag	0 or 1

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

4.54 Expanding Event 302

Applies To: *Grid, TreeView*

Description

If enabled, this event is reported by a *Grid* or a *TreeView* object just before it is about to expand to show additional rows or children of the current item.

In a *Grid*, this occurs when the user clicks the picture or tree line in the row title.

In a *TreeView*, this occurs when the user double-clicks the item label or clicks in the button or on the tree line to the left of the item label.

The default processing for the event is to expand the tree at the corresponding point.

You may disable the expand operation by setting the action code for the event to $\bar{1}$. You may also prevent the expand from occurring by returning 0 from a callback function. You may expand an object dynamically under program control by calling `Expanding` as a method.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Expanding' or 302
[3]	Item number	Integer. The index of the row or item.

4.55 Expose Event 32

Applies To

Description

If enabled, this event is reported when part or all of the object's window is exposed to view. Under normal circumstances, APL repaints the exposed region automatically. However, if you have drawn unnamed graphical objects (which are **not** managed by APL) you should use this event to redraw them when required. Note that APL will itself repaint any **named** objects in the region before reporting the event.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Expose' or 32
[3]	Y	y-position of top-left corner of exposed region
[4]	X	x-position of top-left corner of exposed region
[5]	H	height of exposed region
[6]	W	width of exposed region

This event **cannot** be disabled by setting its action code to $\bar{1}$. Similarly, setting the result of a callback function to 0 has no effect on it.

4.56 FileBoxCancel Event 72

Applies To: *BrowseBox, FileBox*

Description

If enabled, this event is reported when a *FileBox* is closed because the user has pressed the "Cancel" button or closed it.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'FileBoxCancel' or 72
[3]	File name	character vector containing the name of the currently selected file (empty if none)

4.57 FileBoxOK Event 71

Applies To: *BrowseBox, FileBox*

Description

If enabled, this event is reported when a *FileBox* is closed because the user has pressed the "Open" or "Save" button.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'FileBoxOK' or 71
[3]	File name	character vector containing the name of the currently selected file (empty if none)

4.58 FontCancel Event 242

Applies To

Description

If enabled, this event is reported when the user has pressed the *Cancel* button or closed the font selection dialog box that is displayed by the `ChooseFont` method.

The event message reported as the result of `FontDialog`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'FontCancel' or 242

4.59 FontOK Event 241

Applies To

Description

If enabled, this event is reported when the user has pressed the *OK* button in the font selection dialog box that is displayed by the `ChooseFont` method.

The event message reported as the result of `FontDialog`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'FontOK' or 241
[3]	Font	nested vector
[4]	Colour	RGB triplet

The font specification in the 3rd element of the event message is a 7-element nested vector that describes the chosen font. See `Font Object` for further details.

The colour specification in the 4th element of the event message is a 3-element integer vector of RGB values for the colour chosen by the user.

4.60 FrameContextMenu Event 411

Applies To: *Form, SubForm*

Description

If enabled, this event is reported when the user clicks and releases the right mouse button over the non-client area of an object, for example, the title bar in a `Form`.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'FrameContextMenu' or 411
[3]	Y	y-position of the mouse (number)
[4]	X	x-position of the mouse (number)

For further details, see *ContextMenu Event*.

4.61 GesturePan Event 494

Applies To

Description

This event is reported when the user touches one or two fingers on an object and drags them .

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GesturePan' or 494
[3]	Flags	integer which reports the state of the gesture
[4]	Location	2-element integer vector containing the y and x-position respectively of the point at which the gesture applies. These are reported in pixel coordinates relative to the origin (top-left corner) of the object reporting the event.
[5]	Distance	2-element integer vector containing the high and low parts (words) of a 64-bit integer that indicates the distance between the two fingers. This will be (0 0) if only one finger is used.

The Flags parameter [3] which reports the state of the Gesture, is an integer with the value 0, 1 (*GF_BEGIN*), 2 (*GF_INERTIA*), 4 (*GF_END*) or 6 (*GF_END+GF_INERTIA*) with the following meanings:

Name	Value	Description
	0	A gesture is in progress
GF_BEGIN	1	A gesture is starting.
GF_INERTIA	2	A gesture has triggered inertia.
GF_END	4	A gesture has finished.

The term *inertia* refers to built-in Windows processing which provides a standardised user-interface including smooth acceleration and de-acceleration of an object.

When the user first touches an object and begins to drag his finger(s), the object generates a GesturePan event with a `Flags` parameter of 1 (GF_BEGIN). Subsequently, if the user drags the object steadily it generates a series of GesturePan events with a `Flags` parameter of 0. When the user lifts his finger(s) away, the object generates a final GesturePan event, with a `Flags` parameter of 4 (GF_END).

If the user *flicks* an object, the system typically continues to generate GesturePan events after the user has ceased to touch the object. These events are generated in response to the acceleration and deceleration imparted by the *flick*, and the `Flags` parameter for these generated events will be 2 (GF_INERTIA) followed (for the last GesturePan event) by 6 (GF_END+GF_INERTIA).

No other event will be reported between the start and end of a series of GesturePan events.

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

Returning zero from the callback disables any default handling by the operating system.

4.62 GesturePressAndTap Event 497

Applies To

Description

This event is reported when the presses one finger on an object and then taps it with a second finger.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GesturePressAndTap' or 497
[3]	Flags	integer which reports the state of the gesture
[4]	Location	2-element integer vector containing the y and x-position respectively of the point midway between the two fingers. These are reported in pixel coordinates relative to the origin (top-left corner) of the object reporting the event..
[5]	Offset	3-element integer vector whose first element is (currently) 0 and whose second and third elements contain the (y,x) offset of the second finger relative to the first.

The Flags parameter [3] which reports the state of the Gesture, is an integer with the value 0, 1 (*GF_BEGIN*), or 4 (*GF_END*):

Name	Value	Description
	0	A gesture is in progress
<i>GF_BEGIN</i>	1	A gesture is starting.
<i>GF_END</i>	4	A gesture has finished.

When the user taps with his second finger, the object generates a *GesturePressAndTap* event with a *Flags* parameter of 1 (*GF_BEGIN*). Subsequently, until the user removes his first finger, it generates a series of *GesturePressAndTap* events with a *Flags* parameter of 0. When the user lifts his first finger away, the object generates a final *GesturePressAndTap* event, with a *Flags* parameter of 4 (*GF_END*)

No other event will be reported between the start and end of a series of *GesturePressAndTap* events.

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

Returning zero from the callback disables any default handling by the operating system.

4.63 GestureRotate Event 495

Applies To

Description

This event is reported when the user touches two fingers on an object and twists them clockwise or anticlockwise.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GestureRotate' or 495
[3]	Flags	integer which reports the state of the gesture
[4]	Location	2-element integer vector containing the y and x-position respectively of the point midway between the two fingers. These are reported in pixel coordinates relative to the origin of the screen.
[5]	Angle	a scalar number which represents the angle of rotation of the twist measured in radians (0 → 02) from the x-axis in a counter-clockwise direction.

The Flags parameter [3] which reports the state of the Gesture, is an integer with the value 0, 1 (*GF_BEGIN*), or 4 (*GF_END*) with the following meanings:

Name	Value	Description
	0	A gesture is in progress
GF_BEGIN	1	A gesture is starting.
GF_END	4	A gesture has finished.

When the user first touches two fingers on an object and begins to twist, the object generates a *GestureRotate* event with a *Flags* parameter of 1 (*GF_BEGIN*). As the user continues to twist his fingers, the object generates a series of *GestureRotate* events with a *Flags* parameter of 0. When the user lifts one or both fingers away, the object generates a final *GestureRotate* event, with a *Flags* parameter of 4 (*GF_END*).

No other event will be reported between the start and end of a series of *GestureRotate* events.

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

Returning zero from the callback disables any default handling by the operating system.

4.64 GestureTwoFingerTap Event 496

Applies To

Description

This event is reported when the user taps two fingers at the same time on an object

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GestureTwoFingerTap' or 496
[3]	Flags	integer which reports the state of the gesture
[4]	Location	2-element integer vector containing the y and x-position respectively of the point midway between the two fingers. These are reported in pixel coordinates relative to the origin (top-left corner) of the object reporting the event..
[5]	Distance	2-element integer vector containing the high and low parts (words) of a 64-bit integer that indicates the distance between the two fingers.

The Flags parameter [3] which reports the state of the Gesture, is always an integer with the value 5 (*GF_BEGIN+GF_END*).

Name	Value	Description
GF_BEGIN	1	A gesture is starting.
GF_END	4	A gesture has finished.

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

Returning zero from the callback disables any default handling by the operating system.

4.65 GestureZoom Event 493

Applies To

Description

This event is reported when the user touches two fingers on an object and moves them apart or towards each other.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GestureZoom' or 493
[3]	Flags	integer which reports the state of the gesture
[4]	Location	2-element integer vector containing the y and x-position respectively of the centre point of the zoom (the point midway between the two fingers). These are reported in pixel coordinates relative to the origin (top-left corner) of the object reporting the event.
[5]	Distance	2-element integer vector containing the high and low parts (words) of a 64-bit integer that indicates the distance between the two fingers.

The Flags parameter [3] which reports the state of the Gesture, is an integer with the value 0, 1 (*GF_BEGIN*), or 4 (*GF_END*) with the following meanings:

Name	Value	Description
	0	A gesture is in progress
<i>GF_BEGIN</i>	1	A gesture is starting.
<i>GF_END</i>	4	A gesture has finished.

When the user first touches two fingers on an object and begins to move them apart or towards each other, the object generates a *GestureZoom* event with a *Flags* parameter of 1 (*GF_BEGIN*). As the user continues to move the fingers apart or towards each other, the object generates a series of *GestureZoom* events with a *Flags* parameter of 0. When the user lifts one or both fingers away, the object generates a final *GestureZoom* event, with a *Flags* parameter of 4 (*GF_END*).

No other event will be reported between the start and end of a series of *GestureZoom* events.

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

Returning zero from the callback disables any default handling by the operating system.

4.66 GetDayStates Event 266

Applies To: *Calendar*

Description

If enabled, this event is reported when a *Calendar* object requires the APL program to provide *day state* information for the range of dates it is about to display.

The *Calendar* object displays day numbers using either the normal or the bold font attribute. However, it does not store this information beyond the month or months currently displayed.

When the *Calendar* control scrolls (and potentially at other times), it generates a GetDayStates event to ask you (the APL program) to tell it which of the dates that are about to be shown, should be displayed using the bold font attribute.

If you wish any dates to be displayed using the bold font attribute, you **must** attach a callback function to this event which returns day state information in its result.

By default, all dates are displayed using the normal font attribute, so you need only do this if you want any dates highlighted in bold.

You may not disable or nullify the operation that caused GetDayStates to fire by setting the action code for the event to `-1` or by returning 0 from a callback function.

The event message reported as the result of `□DQ`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GetDayStates' or 266
[3]	First Date	an integer (IDN)
[4]	Last Date	an integer (IDN)
[5]	Bold Dates	an integer vector of IDNs.

When the callback function is invoked, the 3rd and 4th elements of the event message contain IDNs for the first and last date in the range of dates that the *Calendar* object is about to display. The 5th element of the event message contains those IDNs *within this*

range of dates that the *Calendar* control already knows are to be displayed using the bold font attribute. This will typically be empty.

The result of your callback function should be the same event message with only the 5th element modified in any way. This should contain the IDNs of the dates (within the range specified by the 3rd and 4th elements) that are to be displayed using the bold font attribute.

EXAMPLE

Suppose that you keep a variable **BOLD_DATES** in the *Calendar* object. This variable is a vector of IDN values that defines those dates that the user has somehow identified as special and that you wish to display in bold. The following callback function could be applied:

```

▼ MSG←DAYSTATES MSG;MASK;[]IO
[1]  A Callback function for the GetDayStates event
[2]  A Object (>MSG) contains a variable BOLD_DATES
[3]  A that defines ALL the IDNs that are to be displayed in bold
[4]  A We need to return only those that fall within the range
[5]  A of dates that are about to be displayed by the Calendar
[6]  []CS>MSG
[7]  []IO←1
[8]  MASK←BOLD_DATES≥3>MSG
[9]  MASK←MASK^BOLD_DATES≤4>MSG
[10] MSG[5]←<MASK/BOLD_DATES
▼

```

You may also set the font attribute for particular days by calling *GetDayStates* as a method.

For example, to set the bold attribute for IDN 36048 (11 September 1998) in a *Calendar* object called 'F.CAL1', you could execute the expression:

```
F.CAL1.GetDayStates 36048 36048 36048
```

To *clear* the bold attribute for the same day:

```
F.CAL1.GetDayStates 36048 36048 0
```

Note that the *Calendar* object will ignore any IDNs you specify that are outside the range of dates that it is currently displaying.

4.67 GetTipText Event 325

Applies To: *ListView*, *TreeView*

Description

If enabled, this event is reported by a *TreeView* or *ListView* object just before it displays a tip for a specific row.

The event message reported as the result of `⌈DQ`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GetTipText' or 325
[3]	Item index	Integer (<code>⌈IO</code> dependent)
[4]	SubItem index	Integer (<code>⌈IO</code> dependent, currently always equal to <code>⌈IO</code>)
[5]	TipText	The text to be displayed.

Modifying and returning the 5th element of the argument to the callback function allows the application to change the displayed tip.

The text can be set to a character array of rank 2 or less.

The default processing for the event is to display the default tip (if there is one).

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

4.68 GotFocus Event 40

Applies To

Description

If enabled, this event is generated when the user has moved the keyboard focus to a new object by clicking the left mouse button, pressing TAB, or using a cursor key.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GotFocus' or 40
[3]	Object name	character vector (name of object which previously had the focus)

The third element (object name) is empty if the focus was obtained from another application window.

The GotFocus event is generated **after** the focus has changed. The default processing is therefore to take no action. However, if you disable the event by setting its action code to `-1`, or inhibit it by returning a 0 from your callback function, the focus is automatically restored to the object (or external application) that had lost it.

4.69 GridCopy Event 191

Applies To: *Grid*

Description

If enabled, this event is reported when the user presses Ctrl+Insert and there are selected cells in the *Grid*. The default action of the event is to copy the contents of the selected block of cells to the clipboard. You may disable this effect entirely by setting the action code of the event to `-1`. You may also disable the copy operation by returning 0 from a callback function.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'GridCopy' or 191
[3]	Start	2-element integer vector or matrix containing the row, column address(es) of the top left cell(s) in the selected block(s)
[4]	End	2-element integer vector or matrix containing the row, column address(es) of the bottom right cell(s) in the selected block
[5]	Values	2-element nested vector. The first element is a matrix containing the values of the selected block(s) of cells. This is the data that will be copied to the clipboard. The second element is a Boolean matrix containing the values of the <i>CellSet</i> property for the selected block(s) of cells.

Note that the values of Start and End are sensitive to the index origin, $\square\text{IO}$.

If more than one block of cells is selected, Start and End are matrices whose rows identify the start and end cells of each of the selected blocks, and Data is the contents of the selected blocks catenated along the appropriate dimension according to their relative positions in the Grid.

You may copy cells under program control by calling GridCopy as a method.

To copy a specific block of cells to the clipboard whether or not they are selected, you must specify the Start and End parameters. For example, the following expression will copy the 3x3 block of cells in the top-left of the Grid ($\square\text{IO}$ is 1) to the clipboard:

```
Gridname.GridCopy (1 1) (3 3)
```

If you omit these parameters, the currently selected block of cells will be copied to the clipboard. If no cells are selected, the entire contents of the Grid will be copied. That is:

```
Gridname.GridCopy ⍉
```

The data copied to the clipboard is registered in Dyalog (APL internal), Wk3 (Lotus), XlTable (Excel) and tab/new-line delimited text formats.

4.70 GridCopyError Event 196

Applies To: *Grid*

Description

If enabled, this event is reported when the user presses Ctrl+Insert and there is more than one block of selected cells in the *Grid* and the blocks are non-conformable. The default action of the event is to generate a Beep. Setting the action code of this event to $\bar{1}$, or returning a 0 from a callback function attached to it, disables the Beep.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'GridCopyError' or 196
[3]		Zilde
[4]		Zilde
[5]	Start	A 2-column integer matrix whose rows identify the address of the first cell (row, column) of each of the selected blocks of cells.
[6]	End	A 2-column integer matrix whose rows identify the address of the last cell (row, column) of each of the selected blocks of cells.

Note that the values of Start and End are sensitive to the index origin, $\square$ IO.

4.71 GridCut Event 190

Applies To: *Grid*

Description

If enabled, this event is reported when the user presses Shift+Delete and there are selected cells in the *Grid*. The default action of the event is to copy the contents of the selected block(s) of cells to the clipboard and then to empty the selected cells. You may disable this effect entirely by setting the action code of the event to $\bar{1}$. You may also disable the cut operation by returning 0 from a callback function.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'GridCut' or 190
[3]	Start	2-element integer vector or matrix containing the row, column address(es) of the top left cell(s) in the selected block(s)
[4]	End	2-element integer vector or matrix containing the row, column address(es) of the bottom right cell in the selected block
[5]	Data	2-element nested vector. The first element is a matrix containing the values of the selected block(s) of cells. This is the data that will be copied to the clipboard. The second element is a Boolean matrix containing the values of the <i>CellSet</i> property for the selected block(s) of cells.

Note that the values of Start and End are sensitive to the index origin, $\square$ IO.

If more than one block of cells is selected, Start and End are matrices whose rows identify the start and end cells of each of the selected blocks, and Data is the contents of the selected blocks catenated along the appropriate dimension according to their relative positions in the Grid.

The data copied to the clipboard is registered in Dyalog (APL internal), Wk3 (Lotus), XITable (Excel) and tab/new-line delimited text formats.

4.72 GridDelete Event 193

Applies To: *Grid*

Description

If enabled, this event is reported when the user presses Delete and there are selected cells in the *Grid*. The default action of the event is to empty the selected cells. You may disable this effect entirely by setting the action code of the event to $\sim$ 1. You may also disable the delete operation by returning 0 from a callback function.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 4-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'GridDelete' or 193
[3]	Start	2-element integer vector or matrix containing the row, column address(es) of the top left cell(s) in the selected block(s)
[4]	End	2-element integer vector or matrix containing the row, column address(es) of the bottom right cell(s) in the selected block(s)

Note that the values of Start and End are sensitive to the index origin, $\square$ IO.

If more than one block of cells is selected, Start and End are matrices whose rows identify the start and end cells of each of the selected blocks.

4.73 GridDropSel Event 195

Applies To: *Grid*

Description

If enabled, this event is reported when the user drag/drops a selected block of cells in the *Grid*. The default action is that the contents of the selected cells replace the values in the block of cells onto which they are dropped and this block now becomes selected.

You may disable the drag/drop facility entirely by setting the action code of the event to `-1`. You may also disable an individual drag/drop operation by returning 0 from a callback function.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GridDropSel' or 195
[3]	Start	2-element integer vector containing the row, column address of the top left cell in the selected block
[4]	Size	2-element integer vector containing the number of rows and columns in the selected block
[5]	Target	2-element integer vector containing the row/column address of the top left cell onto which the selected block is being dropped
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down
[7]	Undo flag	0 or 1
[8]	Values	Matrix containing the values of the selected block of cells. This is the data that will replace the values in the target cells.
[9]	CellSet flags	Boolean Matrix containing the values of the <i>CellSet</i> property for the selected block of cells. This will replace the values of the <i>CellSet</i> property of the target cells.

The shift state in element 6 is intended to allow the APL programmer to implement an *insert* operation instead of a *copy* or *move* operation if required.

You may copy the contents of one block of cells to another by calling *GridDropSel* as a method. If so, you need only specify the *Start*, *Size* and *Target* parameters. Note that the result block becomes selected.

The Undo flag is always 1 if the event was generated by the user.

4.74 GridKeyPress Event 24

Applies To: *Grid*

Description

If enabled, this event is generated when the user presses and releases a key in a *Grid* cell.

The *GridKeyPress* is reported on the *Grid*, *after* the *KeyPress* event, which is reported on the *Input* object associated with the current cell.

The event message reported as the result of `GridKeyPress`, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GridKeyPress' or 24
[3]	Input Code	character scalar or vector
[4]	ASCII code	integer scalar
[5]	Key Number	integer scalar
[6]	Shift State	integer scalar
[7]	Input Object	ref or character vector

For a full description of elements [3-6], see *KeyPress* event.

The 7th element of the event message contains either a reference to, or the name of the *Input* object associated with the current cell and on which the corresponding *KeyPress* event has been reported.

The default action of the *GridKeyPress* event is to pass a *KeyPress* event message back to the appropriate *Input* object to be actioned. The *GridKeyPress* is reported on the *Grid*, after the *KeyPress* event on the *Input* object. If a callback on the *Input* object's *KeyPress* event returns 0 or if a callback on *GridKeyPress* returns 0, the keystroke will be ignored.

4.75 GridPaste Event 192

Applies To: *Grid*

Description

If enabled, this event is reported when the user presses Shift+Insert and there is data in the clipboard that is in a suitable format for the *Grid*. The default action of the event is to copy the contents of the clipboard into the currently selected block of cells, or, if no cells are selected, into the block of cells starting at the current cell (CurCell). Note that if there is a selected range of cells and the shape of the data being pasted does not exactly match the size of the selected range, the system generates a *GridPasteError* event in addition to the GridPaste event.

You may disable the paste facility entirely by setting the action code of the event to -1. You may also disable an individual paste operation by returning 0 from a callback function.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GridPaste' or 192
[3]	Values	New values (taken from the clipboard) which are to replace the existing values of the block of cells defined by Start and End.
[4]	CellSet flags	Boolean Matrix containing the new values of the <i>CellSet</i> property for the block of cells defined by Start and End.
[5]	Start	2-element integer vector containing the row, column address of the top left cell the selected block. If there is no selection, this is the address of the current cell(CurCell).
[6]	End	2-element integer vector containing the row, column address of the bottom right cell in the selected block. If there is no selection, this is the address of the bottom right cell of the block starting at the current cell that will be overwritten

You can replace the contents of a contiguous block of cells with the data in the clipboard, or with an arbitrary matrix of values, by calling GridPaste as a method.

If you call GridPaste with an argument of 0, the data is taken from the clipboard; otherwise the data to be pasted is specified by the *Values* and *CellSet flags* parameters.. If you omit *Start*, data is pasted into the currently selected range of cells. If there are no cells selected, data is pasted starting at the current cell (CurCell). In either case, the block of replaced cells becomes selected.

4.76 GridPasteError Event 194

Applies To: *Grid*

Description

If enabled, this event is reported when the user presses Shift+Insert and there is data in the clipboard, but the system is unable to paste the data into the *Grid*. This occurs if there is a currently selected block of cells whose shape does not match the shape of the data in the clipboard. It also occurs if there is no selected block of cells, and pasting the data in starting at the current cell (CurCell) would overflow the *Grid*. Setting the action code of this event to -1, or returning a 0 from a callback function attached to it, has no effect.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'GridPasteError' or 194
[3]	Values	Contents of the clipboard.
[4]	CellSet flags	Boolean array indicating which elements of the clipboard data are empty.
[5]	Start	2-element integer vector containing the row, column address of the top left cell in the selected block. If there is no selection, this is the address of the current cell (CurCell).
[6]	End	2-element integer vector containing the row, column address of the bottom right cell in the selected block. If there is no selection, this is the address of the bottom right cell of the block starting at the current cell that will be overwritten
[7]	Error Number	4 (RANK ERROR) or 5 (LENGTH ERROR)

4.77 GridSelect Event 165

Applies To: *Grid*

Description

If enabled, this event is reported when the user performs or cancels the selection of a block of cells in a *Grid* object. This event is reported after the selection has changed. Setting its action code to `~1` has no effect and the result of a callback function cannot be used to alter the selection that has been made. You may however control the user's ability to make selections using the *CellSelect* property.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GridSelect' or 165
[3]	Start	2-element integer vector or matrix containing the row, column address(es) of the top left cell(s) in the selected block(s)
[4]	End	2-element integer vector or matrix containing the row, column address(es) of the bottom right cell(s) in the selected block

Note that the values of Start and End are sensitive to the index origin, `⍋IO`.

If the selection is made with the mouse, the GridSelect event is reported when the left mouse button is released. If the selection is made using the cursor keys, the GridSelect event is reported when the Shift key is released.

The GridSelect event is also generated when the current selection is cancelled by clicking on a cell with the mouse or by pressing a cursor key.

4.78 HScroll Event 39

Applies To: *Form*, *SubForm*

Description

If enabled, this event is generated when the user attempts to move the thumb in a horizontal scrollbar in a *Form* or *SubForm*. This event occurs only in a *Form* whose *HScroll* property is set to `~1` and is distinct from the Scroll event that is generated by a *Scroll* object. The event may be generated in one of three ways:

- dragging the thumb.
- clicking in one of the "arrow" buttons situated at the ends of the scrollbar. This is termed a small change, the size of which is defined by *Step*[3].
- clicking in the body of the scrollbar. This is termed a large change, the size of which is defined by *Step*[4].

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 4-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'HScroll' or 39
[3]	Scroll Type	numeric
[4]	Position	numeric

The value of Scroll Type is 0 (drag), 1 or $\bar{1}$ (small change) or 2 or $\bar{2}$ (large change). The sign indicates the direction.

The value of Position is the new (requested) position of the thumb. Notice however, that the event is generated **before** the thumb is actually moved. If your callback function returns a scalar 0, the position of the thumb will remain unaltered.

4.79 HTTPRequest Event 840

Applies To: *HTMLRenderer*

Description

An HTTPRequest event is raised whenever the *HTMLRenderer* requests a url from the workspace. See *InterceptedURLs* property. The request could be generated by a form submission, clicking on a hyperlink, an AJAX request or a link to a resource like a style sheet, image or JavaScript file. An HTTPRequest event is also raised when the *HTMLRenderer* is initialised and both *HTML* and *URL* are empty.

The callback function must "fill in the blanks" in the event message and return the modified event message as its result.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is an 11-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'HTTPRequest' or 840
[3]		'ProcessRequest' (unused)
[4]	Handle	Initially 0, must be set to 1.
[5]	Status	Integer HTTP status code (initially 0).
[6]	Message	Character vector containing the HTTP status message (initially empty).
[7]	MIME	Character vector containing the MIME type (initially empty). See below.
[8]	URL	Character vector containing the requested URL.
[9]	Headers	Character vector containing the HTTP Request headers (initially empty).
[10]	Body	Character vector containing the HTTP Request body (initially empty).
[11]	Method	Character vector containing the HTTP method, for example, 'GET' or 'POST' .

To process the request, the callback function should return the message with only the following items changed. Note that only elements [4 5 6 10] are always required, and it is important to set element [4] to 1.

[4]	Handle	1
[5]	Status	Success is indicated by 200.
[6]	Message	Success is indicated by 'OK' .
[7]	MIME	Defaults to 'text/html' and need be specified only if the response (Body) is not a character vecttor containing HTML.
[9]	Headers	Not normally required.
[10]	Body	This may be a character vector or an integer numeric vector with values in the ranges ~128-127 or 0-255. Note that a character vector will be prepended with a UTF-8 byte order mark (BOM). A numeric vector requires that [7] is an appropriate MIME type (for example, 'image/png')

MIME types include:

- text/html
- text/css
- text/plain
- text/csv
- application/javascript
- application/xml
- application/json

For a complete list of media/MIME types, see:

<https://www.iana.org/assignments/media-types/media-types.xhtml>:

EXAMPLE

```
<!DOCTYPE html>
<html>
<head>
<Title>HTTPRequest Example</Title>
</head>
<body><h1>Simple Form</h1>

<form action="Hello">
  First name:<br>
  <input type="text" name="firstname" value="Mickey">
  <br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse">
  <br><br>
  <input type="submit" value="Submit">
</form>

<p>When you click the "Submit" button, the HTMLRenderer
  will fire an HTTPRequest event.</p>

</body>
</html>
```



```

▼ msg←SayHello msg;url;names;first;last;response
[1]   url←8>msg
[2]   'Requested URL is: ',url
[3]   names←(url←'?&')<url
[4]   first last←{(ω1='')↓ω}names
[5]   response←'!DOCTYPE html><html><head>'
[6]   response,←'<Title>Hi Folks</Title>'
[7]   response,←'</head><body>'
[8]   response,←'<h1 align="center">Hello '
[9]   response,←first,' ',last,'</h1>'
[10]  response,←'</body></html>'
[11]  msg[4 5 6 10]←1 200 'OK' response

```

```

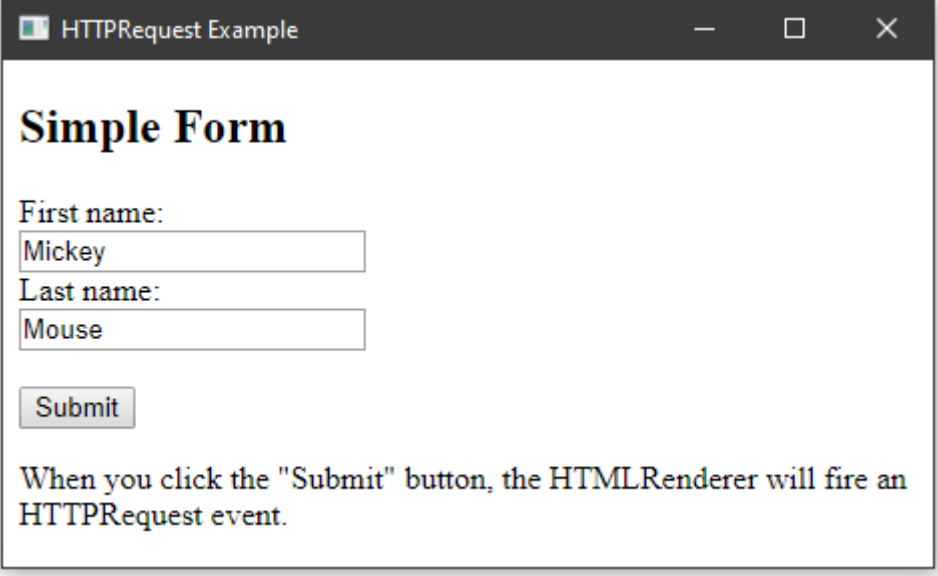
▼ run
[1]   'hr'⌈WC'HTMLRenderer'
[2]   hr.Size←25 25
[3]   hr.HTML←html
[4]   hr.onHTTPRequest←'SayHello'

```

```

run
Requested URL is: http://dyalog_root/Hello?firstname=Mickey&lastname=Mou
se

```



HTTPRequest Example

Simple Form

First name:

Last name:

When you click the "Submit" button, the HTMLRenderer will fire an HTTPRequest event.



4.80 HThumbDrag Event 442

Applies To: *Form, SubForm*

Description

If enabled, this event is generated when the user attempts to drag the thumb in a horizontal scrollbar in a *Form* or *SubForm*. This event occurs only in a *Form* or *SubForm* whose *HScroll* property is set to `-1` and is distinct from the *Scroll* event that is generated by a *Scroll* object.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'HThumbDrag' or 442
[3]	Position	numeric

The value of *Position* is the new (requested) position of the Thumb. Setting the action code of this event to `-1`, or returning a 0 from a callback function attached to it, has no effect.

4.81 Help Event 400

Applies To

Description

If enabled, this event is reported when the user clicks on an object which has a callback defined for this event, the user having previously clicked on the Question (?) button in the title bar of the parent Form. The presence of the Question (?) button is determined by the value of the *HelpButton* property.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' Help or 400
[3]	Y-coordinate	Number
[4]	X-coordinate	Number

The y and x-coordinates refer to the position of the mouse pointer in the object which was clicked on, and are reported in the coordinate system of that object.

4.82 HideComment Event 224

Applies To: Grid

Description

If enabled, a HideComment event is generated just before a comment window is hidden as a result of the user moving the mouse-pointer away from a commented cell.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' HideComment ' or 224
[3]	Row	integer
[4]	Column	integer

You may prevent the comment from being hidden by returning 0 as the result of a callback function.

Note that if the comment window relates to a row or column *title*, the value reported in element [3] or [4] of the event message is `⍒1`.

Invoked as a method, HideComment is used to hide a comment that has previously been displayed by ShowComment. For example, the following expression hides the comment associated with the cell at row 2, column 1.

```
F.G.HideComment 2 1
```

If HideComment is called with an argument of `⍬`, all comments are hidden.

4.83 Idle Event 130

Applies To: *Root*

Description

If enabled, this event is generated whenever APL looks to see if there is an event on the queue and finds it empty. Its purpose is to allow an application to perform some background processing when the user is not doing anything. It is unwise to use this event directly from the Session as it will occur repeatedly and may lock you out.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function is a 2-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'Idle' or 130

4.84 IndexChanged Event 210

Applies To: *Grid*

Description

If enabled, this event is reported when the value of the Index property of a Grid has changed as a result of user interaction. The event is reported *after* the Grid has been scrolled. You may not modify or nullify the operation with a 0-return callback and you may not call IndexChanged as a method or generate this event using `⍋NQ`. To cause a Grid to scroll, use `⍋WS` to set its Index property.

The event message reported as the result of `WM_NOTIFY`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'IndexChanged' or 210
[3]	Row	Integer.
[4]	Column	Integer.

4.85 ItemDbClick Event 342

Applies To: *ListView*, *TreeView*

Description

If enabled, this event is reported when the user double-clicks a mouse button when the mouse pointer is over an item in a *ListView* or *TreeView* object. This event is reported for information only and may not be controlled in any way using a callback function. Generating the event with `SendMessage`, or calling it as a method, has no effect.

The event message reported as the result of `WM_NOTIFY`, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ItemDbClick' or 342
[3]	Item number	Integer. The index of the item.
[4]	Mouse button	Integer.
[5]	Shift state	Integer. Sum of 1=shift key, 2=Ctrl key, 4=Alt key
[6]	Position	Integer. Indicates the position of the mouse-pointer within the item. It is either 2 (over the icon), 4 (over the label), 8 (over the line), or 16 (over the symbol).

4.86 ItemDown Event 340

Applies To: *ListView*, *TreeView*

Description

If enabled, this event is reported when the user depresses a mouse button when the mouse pointer is over an item in a *ListView* or *TreeView* object. This event is reported for information only and may not be controlled in any way using a callback function. Generating the event with `SendMessage`, or calling it as a method, has no effect.

The event message reported as the result of `SendMessage`, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ItemDown' or 340
[3]	Item number	Integer. The index of the item.
[4]	Mouse button	Integer.
[5]	Shift state	Integer. Sum of 1=shift key, 2=Ctrl key, 4=Alt key
[6]	Position	Integer. Indicates the position of the mouse-pointer within the item. It is either 2 (over the icon), 4 (over the label), 8 (over the line), 16 (over the symbol) or 32 (to the right of the label).

4.87 ItemUp Event 341

Applies To: *ListView*, *TreeView*

Description

If enabled, this event is reported when the user releases a mouse button when the mouse pointer is over an item in a *ListView* or *TreeView* object. This event is reported for information only and may not be controlled in any way using a callback function. Generating the event with `SendMessage`, or calling it as a method, has no effect.

The event message reported as the result of `SendMessage`, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ItemUp' or 341
[3]	Item number	Integer. The index of the item.
[4]	Mouse button	Integer.
[5]	Shift state	Integer. Sum of 1=shift key, 2=Ctrl key, 4=Alt key
[6]	Position	Integer. Indicates the position of the mouse-pointer within the item. It is either 2 (over the icon), 4 (over the label), 8 (over the line), or 16 (over the symbol).

4.88 KeyError Event 23

Applies To: *ButtonEdit, Edit, Spinner*

Description

If enabled, this event is generated when the user presses and releases a key on the keyboard that is invalid for the FieldType of the object and has been ignored. This event is reported for information only and you may not disable it or modify it in any way.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'KeyError' or 23
[3]	Character	character scalar
[4]	Character code	integer scalar
[5]	Key Number	integer scalar
[6]	Shift state	integer scalar

In the Classic Edition, the resolution of the keystroke to a character (in □AV) is performed using the Input Translate Table. In the Unicode Edition, the resolution is performed by the Operating System.

In the Unicode Edition, the Character Code is the Unicode code point of the character that the user entered. In the Classic Edition, it is a number in the range 0-255 which specifies the ASCII character that would normally be generated by the keystroke, and is independent of the Input Translate Table. If there is no corresponding ASCII character, the ASCII code reported is 0.

The key number is the physical key number reported by Windows when the key is pressed.

The Shift State indicates which (if any) of the Shift, Ctrl and Alt keys are down at the same time as the key is pressed. It is the sum of the following numbers :

Thus a Shift State of 3 indicates that the user has pressed the key in conjunction with both the Shift and Ctrl keys. A Shift State of 0 indicates that the user pressed the key on its own.

4.89 KeyPress Event 22

Applies To

Description

If enabled, this event is generated when the user presses and releases a key on the keyboard. It is reported for whichever object has the keyboard focus at the time.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'KeyPress' or 22
[3]	Input Code	character scalar or vector
[4]	Character code	integer scalar
[5]	Key Number	integer scalar
[6]	Shift state	integer scalar

If the keystroke resolves to a character, the Input Code is a character scalar.

If the keystroke resolves to a command recognised by Dyalog APL, such as UC (Up Cursor) or ER (Enter) the Input Code contains the corresponding 2-element character vector.

In the Classic Edition, the resolution of the keystroke to a character (in □AV) or to a command, is performed using the Input Translate Table.

In the Unicode Edition, the resolution is performed by the Operating System. However, if the keystroke resolves to a navigation or control key (such as Cursor Up or Enter), the same 2-character "command" is reported. Note however that commands that are purely internal to Dyalog APL (such as Trace, commonly Ctrl+Enter) are not reported as such and the Input Code will be empty.

In the Unicode Edition, the Character Code is the Unicode code point of the character that the user entered. In the Classic Edition, it is a number in the range 0-255 which specifies the ASCII character that would normally be generated by the keystroke, and is independent of the Input Translate Table. If there is no corresponding ASCII character, the ASCII code reported is 0.

The key number is the physical key number reported by Windows when the key is pressed.

The Shift State indicates which (if any) of the Shift, Ctrl and Alt keys are down at the same time as the key is pressed. It is the sum of the following numbers :

1	Shift key down
2	Control key down
4	Alt key down

Thus a Shift State of 3 indicates that the user has pressed the key in conjunction with both the Shift and Ctrl keys. A Shift State of 0 indicates that the user pressed the key on its own.

EXAMPLE

```

▼ Key;Form1
[1]   'Form1'□WC'Form'('Event' 'KeyPress' 'Keycb')
[2]   □DQ'Form1'
▼

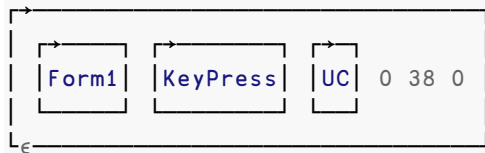
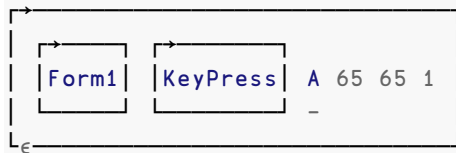
▼ Keycb msg
[1]   DISPLAY msg
▼

```

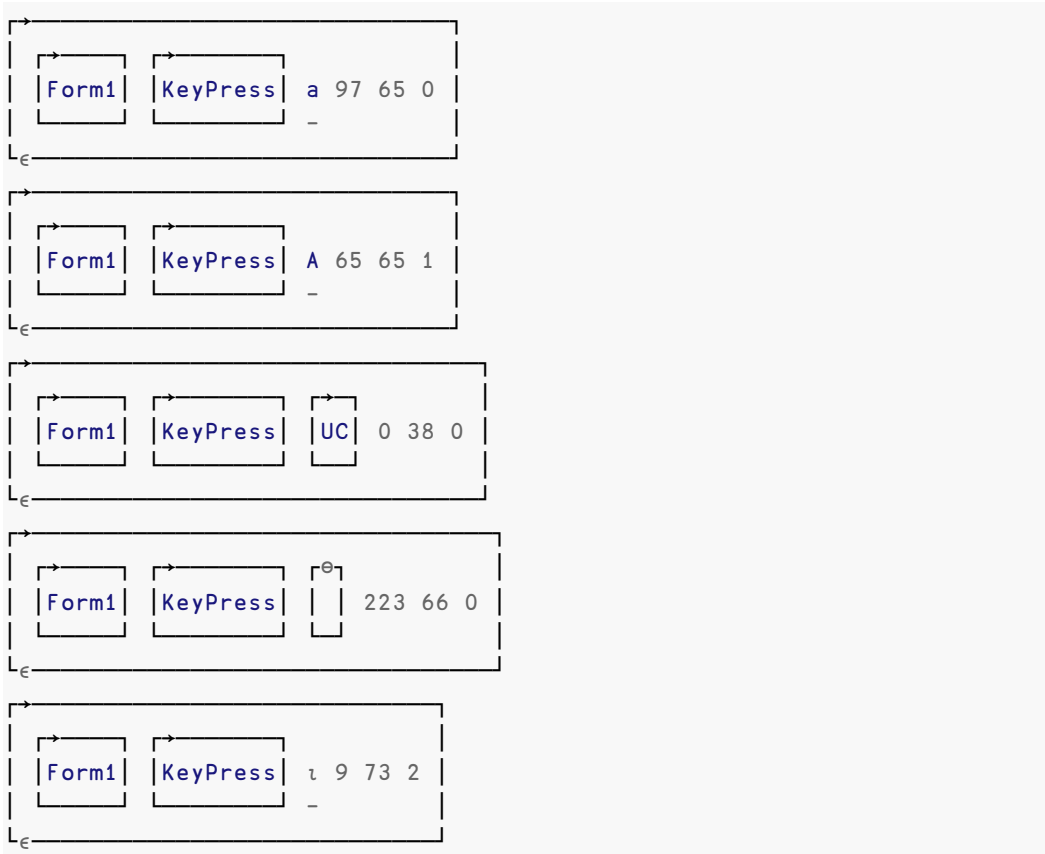
On running function Key, the following output will be displayed as a result of the user pressing the following 5 keys in succession:

1. "a"
2. Shift+"a"
3. Cursor Up
4. β ("b" using a Greek keyboard)
5. ι (Ctrl+"i" using a UK APL keyboard)

Unicode Edition



Classic Edition



4.90 LoadEnd Event 836

Applies To: *HTMLRenderer*

Description

A LoadEnd event is raised when a particular frame has finished loading. Multiple frames may be loading at the same time. Sub-frames may start or continue to load even after the main frame has finished loading.

A common technique is to wait for the main frame to finish loading before further interaction with the HTMLRenderer instance. In this case, you should set up an event handler on the LoadEnd event and check the 4th element which indicates if the loaded frame is the main frame.

You may use the *IsLoading* property to check if the HTMLRenderer is still loading.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'LoadEnd' or 836
[3]	url	The URL of the loaded frame
[4]	Flag	1 if the loaded frame is the "main" frame, 0 otherwise
[5]	Code	The HTTP status code as a result of loading the frame

4.91 Locator Event 80

Applies To: *Locator*

Description

If enabled, this event is generated when the user releases a mouse button, or presses any key (other than a cursor movement key) during a *Dyalog APL Language: Dq* on a *Locator* object.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 9-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Locator' or 80
[3]	Y	y-position of <i>Locator</i> after <i>Dyalog APL Language: Dq</i>
[4]	X	x-position of <i>Locator</i> after <i>Dyalog APL Language: Dq</i>
[5]	H	height of <i>Locator</i> after <i>Dyalog APL Language: Dq</i>
[6]	W	width of <i>Locator</i> after <i>Dyalog APL Language: Dq</i>
[7]	Mouse Button	number of the button which was released (0 if keystroke)
[8]	Keystroke	character scalar or vector containing the "Input Code" for the key that terminated the operation
[9]	Shift state	integer scalar

4.92 LostFocus Event 41

Applies To

Description

If enabled, this event is generated when the user transfers the keyboard focus away from the object in question.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'LostFocus' or 41
[3]	Object name	character vector (name of object that has received the focus)

If the focus is transferred to a window that is not part of the Dyalog APL GUI Interface, the third element is an empty vector.

The LostFocus event is generated **after** the focus has changed. The default processing is therefore to take no action. However, if you inhibit the event by returning a 0 from your callback function, the focus is automatically restored to the object that had lost it.

4.93 MDIActivate Event 42

Applies To: SubForm

Description

This event is generated when the user activates a particular *SubForm* that is the child of an *MDIClient*. This occurs when the user clicks the left mouse button in the *SubForm* or selects it from the menu nominated for this purpose (see *MDIMenu* property). You may also call MDIActivate as a method.

Note that this event is reported after the action has taken place and cannot be disabled by returning 0 from a callback function or by setting its action code to -1.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'MDIActivate' or 42
[3]	Object name	character vector

Note that the 3rd element of the event message is either an empty vector or contains the name of the *SubForm* that was previously the active one in the same *MDIClient*.

4.94 MDIDeactivate Event 43

Applies To: *SubForm*

Description

This event is generated when the user activates a different *SubForm* that is the child of an *MDIClient*, thereby de-activating the current one which causes this event. This occurs when the user clicks the left mouse button in another *SubForm* or selects it from the menu nominated for this purpose (see *MDIMenu* property). You may also call *MDIDeactivate* as a method.

Note that this event is reported after the action has taken place and cannot be disabled by returning 0 from a callback function or by setting its action code to -1.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'MDIDeactivate' or 43
[3]	Object name	character vector

Note that the 3rd element of the event message contains the name of the *SubForm* that has now been made the active one in the same *MDIClient*.

4.95 MouseDbClick Event 5

Applies To

Description

If enabled, this event is reported when the user presses and then releases a mouse button twice within a short space of time. The duration of this time is set through the Windows Control Panel.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'MouseDownClick' or 5
[3]	Y	y-position of mouse (number)
[4]	X	x-position of mouse (number)
[5]	Button	button double clicked (number) 1 = left button 2 = right button 4 = middle button
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down

In a graphical object (*Circle*, *Ellipse*, *Image*, *Marker*, *Poly* and *Rect*), the position of the mouse is reported relative to the top-left corner of its bounding rectangle.

If you enable *MouseDown* and *MouseUp* events in addition to *MouseDownClick* events, double-clicking a mouse button will generate the following sequence of events :

1. *MouseDown*
2. *MouseUp*
3. *MouseDownClick*
4. *MouseUp*

4.96 MouseDown Event 1

Applies To

Description

If enabled, this event is reported when the user presses one of the mouse buttons. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'MouseDown ' or 1
[3]	Y	y-position of mouse (number)
[4]	X	x-position of mouse (number)
[5]	Button	button pressed (number) 1 = left button 2 = right button 4 = middle button
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down

If you enable this event it is advisable that you ALSO enable *MouseUp* events. Otherwise, the slight delay in running your callback function will cause the down and up sequence to be **reversed**.

In a graphical object (*Circle*, *Ellipse*, *Image*, *Marker*, *Poly* and *Rect*), the position of the mouse is reported relative to the top-left corner of its bounding rectangle.

4.97 MouseEnter Event 6

Applies To

Description

If enabled, this event is reported when the user moves the mouse pointer into (over) an object. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'MouseEnter ' or 6
[3]	Object name	character vector (name of previous object)

This event is generated when the user moves the mouse pointer across the boundary and into an object. The first element of the event message is the name of the object over which the mouse pointer now resides. The 3rd element of the event message

contains the name of the object that was previously under the mouse pointer, or is an empty vector if the mouse pointer was not previously over a Dyalog APL/W object.

4.98 MouseLeave Event 7

Applies To

Description

If enabled, this event is reported when the user moves the mouse pointer out of an object. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'MouseLeave' or 7
[3]	Object name	character vector (name of new object)

This event is generated when the user moves the mouse pointer across the boundary and away from an object. The first element of the event message contains the name of the object that previously contained the mouse pointer and which generated the event when it crossed its boundary. The third element contains the name of the object which now contains the mouse pointer or is an empty vector if the mouse pointer is not now over a Dyalog APL/W object.

4.99 MouseMove Event 3

Applies To

Description

If enabled, this event is reported when the user moves the mouse. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'MouseMove' or 3
[3]	Y	y-position of mouse (number)
[4]	X	x-position of mouse (number)
[5]	Button	button released (number) 1 = left button 2 = right button 4 = middle button
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down

In a graphical object (*Circle*, *Ellipse*, *Image*, *Marker*, *Poly* and *Rect*), the position of the mouse is reported relative to the top-left corner of its bounding rectangle.

Note that rapid movement of the mouse will not necessarily cause an overwhelming number of MouseMove events to be reported, as several small movements are automatically combined into one large one.

4.100 MouseUp Event 2

Applies To

Description

If enabled, this event is reported when the user releases one of the mouse buttons. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'MouseUp' or 2
[3]	Y	y-position of mouse (number)
[4]	X	x-position of mouse (number)
[5]	Button	button released (number) 1 = left button 2 = right button 4 = middle button
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down

In a graphical object (*Circle*, *Ellipse*, *Image*, *Marker*, *Poly* and *Rect*), the position of the mouse is reported relative to the top-left corner of its bounding rectangle.

4.101 MouseWheel Event 8

Applies To

Description

If enabled, this event is reported when the user rotates the mouse wheel.

The event message reported as the result of `OnMouseWheel`, or supplied as the right argument to your callback function, is a 9-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'MouseWheel ' or 8
[3]	Y	y-position of mouse (number)
[4]	X	x-position of mouse (number)
[5]	Button	button pressed 1 = left button 2 = right button 4 = middle button
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down
[7]	Delta	integer
[8]	Lines	integer
[9]	Wheel Delta	integer

The value of *Delta* indicates the distance that the wheel is rotated, expressed in multiples or divisions of *Wheel Delta*. A positive value indicates that the wheel was rotated forward, away from the user; a negative value indicates that the wheel was rotated backward, toward the user.

Lines specifies the number of lines to scroll when the wheel is rotated by 1 *Mouse Delta* unit. A value of -1 indicates that a whole screen is to be scrolled. These values are defined by the user's preferences (*Control Panel/Mouse*).

4.102 MsgBtn1 Event 61

Applies To: *MsgBox*

Description

If enabled, this event is reported when the user responds to a *MsgBox* object by clicking its first (leftmost) button. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'MsgBtn1' or 61

4.103 MsgBtn2 Event 62

Applies To: *MsgBox*

Description

If enabled, this event is reported when the user responds to a *MsgBox* object by clicking its second (from the left) button. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'MsgBtn2' or 62

4.104 MsgBtn3 Event 63

Applies To: *MsgBox*

Description

If enabled, this event is reported when the user responds to a *MsgBox* object by clicking its third (from the left) button. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'MsgBtn3' or 63

4.105 PageActivate Event 360

Applies To: *PropertyPage*

Description

If enabled, this event is reported when the user switches from one *PropertyPage* to another in a *PropertySheet* object. This event is reported by the new page *after* the page change has occurred and the page change may not be disabled by a callback function.

The event message reported as the result of `MsgBox`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'PageActivate' or 360

You may select a particular page by calling `PageActivate` as a method, or by setting the *PageActive* or *PageActiveObject* property of the *PropertySheet*.

4.106 PageApply Event 350

Applies To: *PropertyPage*

Description

If enabled, this event is reported when the user clicks the Apply button in a *PropertySheet*. Note however, that the event is actually reported by each of its *PropertyPage* objects whose *Changed* property is currently 1, that is, the event is reported by each of the pages that the user has changed.

The default processing for this event is to set the *Changed* property of the *PropertyPage* to 0. If you disable the event or return a 0 from a callback function, the *Changed* property is not reset. Note that the Apply button in a *PropertySheet* is active if the value of the *Changed* property of *any* of the *PropertyPage* objects is 1.

The event message reported as the result of `□□□`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' PageAppLy ' or 350

4.107 PageBack Event 353

Applies To: *PropertyPage*

Description

If enabled, this event is reported when the user switches from one *PropertyPage* to another in a Wizard *PropertySheet* object by clicking its Back button. This event is reported by the old page *after* the page change has occurred and the page change may not be disabled by a callback function.

The event message reported as the result of `□□□`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' PageBack ' or 353

4.108 PageCancel Event 351

Applies To: *PropertyPage*

Description

If enabled, this event is reported when the user presses the Cancel button in a *PropertySheet* object and is reported by the current *PropertyPage*. This event is reported for information only and may not be disabled by a callback function. However, the operation will also generate a Close event reported by the *PropertySheet* itself that may be disabled by a callback.

The event message reported as the result of `□□□`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'PageCancel ' or 351

4.109 PageChanged Event 356

Applies To: *PropertyPage*

Description

If enabled, this event is reported when the *Changed* property of a *PropertyPage* is altered by user action. It is *not* reported if you reset the *Changed* property using `WS`.

The *Changed* property is reset by two separate user actions. It is set to 1 when the user alters any of the controls on the *PropertyPage*. It is reset to 0 when the user clicks the Apply button, although this action may be disabled by a callback function on the *PageApply* event.

The *PageChanged* event is reported for information only and may not itself be disabled or affected by a callback function.

The event message reported as the result of `DDQ`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'PageChanged ' or 356
[3]	Changed value	New value for the <i>Changed</i> property (0 or 1).

4.110 PageDeactivate Event 361

Applies To: *PropertyPage*

Description

If enabled, this event is reported when the user switches from one *PropertyPage* to another in a *PropertySheet* object. This event is reported by the old page *after* the page change has occurred and the page change may not be disabled by a callback function.

The event message reported as the result of `DDQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'PageDeactivate' or 361

4.111 PageFinish Event 355

Applies To: *PropertyPage*

Description

If enabled, this event is reported when the user clicks the Finish button in a Wizard *PropertySheet*. This event is reported by current (last) *PropertyPage*. The event is reported for information only and cannot be affected by a callback function.

The event message reported as the result of `DDQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'PageFinish' or 355

4.112 PageHelp Event 352

Applies To: *PropertyPage*

Description

If enabled, this event is reported when the user clicks the Help button in a Wizard *PropertySheet*. This event is reported by current *PropertyPage*. The event is reported for information only and cannot be affected by a callback function.

The event message reported as the result of `DDQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'PageHelp' or 352

4.113 PageNext Event 354

Applies To: *PropertyPage*

Description

If enabled, this event is reported when the user switches from one *PropertyPage* to another in a Wizard *PropertySheet* object by clicking its Next button. This event is reported by the old page *after* the page change has occurred and the page change may not be disabled by a callback function.

The event message reported as the result of `WM_NOTIFY`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' PageNext ' or 354

4.114 PreCreate Event 534

Applies To: *ActiveXControl*

Description

If enabled, this event is reported when an instance of an *ActiveXControl* is created. The PreCreate event is generated at the point the *instance* is made.

An *ActiveXControl* also generates a *Create* event, which occurs *after* the PreCreate event at the point when the host application requires the instance to appear visually.

Note that at the time that PreCreate is generated, the *ActiveXControl* does not have a window.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to `-1` or by returning 0 from a callback function.

The event message reported as the result of `WM_NOTIFY`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' PreCreate ' or 534

4.115 Protected Event 470

Applies To: *RichEdit*

Description

If enabled, this event is reported when the user attempts to alter protected text in a RichEdit. See *CharFormat* property.

The event message reported as the result of `WM_`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Protected' or 470

4.116 Retracting Event 304

Applies To: *Grid*, *TreeView*

Description

If enabled, this event is reported by a *Grid* or a *TreeView* object just before it is about to retract to hide the children of the current item.

In a *Grid*, this occurs when the user clicks the picture or tree line in the row title.

In a *TreeView*, this occurs when the user double-clicks the item label or clicks in the button or on the tree line to the left of the item label, when the item is in its expanded state.

The default processing for the event is to retract the tree at the corresponding point.

You may disable the retract operation by setting the action code for the event to `-1`. You may also prevent the retraction from occurring by returning 0 from a callback function. You may retract *Grid* or a *TreeView* dynamically under program control by calling *Retracting* as a method.

The event message reported as the result of `WM_`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Retracting' or 304
[3]	Item number	Integer. The index of the item.

4.117 Scroll Event 37

Applies To: *Scroll*, *TrackBar*

Description

If enabled, this event is generated when the user attempts to move the thumb in a scrollbar. This can be done in one of three ways:

1. dragging the thumb.
2. clicking in one of the "arrow" buttons situated at the ends of the scrollbar. This is termed a small change, the size of which is defined by *Step[1]*.
3. clicking in the body of the scrollbar. This is termed a large change, the size of which is defined by *Step[2]*.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Scroll' or 37
[3]	Scroll Type	numeric
[4]	Position	numeric

The value of Scroll Type is 0 (drag), 1 or $\bar{1}$ (small change) or 2 or $\bar{2}$ (large change). The sign indicates the direction.

The value of Position is the new (requested) position of the thumb. Notice however, that the event is generated **before** the thumb is actually moved. If your callback function returns a scalar 0, the position of the thumb will remain unaltered.

4.118 SelDateChange Event 265

Applies To: *Calendar*

Description

If enabled, this event is reported when the user changes the date, or range of dates, that is selected in a *Calendar* object. This event is also reported when the *Calendar* object is scrolled and the selection changes automatically to another month.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to `-1` or by returning 0 from a callback function.

The event message reported as the result of `DQ`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'SelDateChange' or 265
[3]	First Date	an integer (IDN)
[4]	Last Date	an integer (IDN)

4.119 Select Event 30

Applies To

Description

For a *Button* with *Style* 'Push' this event is generated when the user "pushes" the button. This can be done by clicking the left mouse button, or by pressing the Enter key or the space bar when the *Button* has the focus. The Select event can also be generated when the *Button* does not have the focus, by pressing the Enter key when its *Default* property is 1 or by pressing the ESC key when its *Cancel* property is 1.

For a *Button* with *Style* 'Radio' or 'Check' this event is generated when the user toggles the button from one state to another. This can be achieved by clicking the left mouse button or by pressing the space bar when the *Button* has the focus.

For a *Combo* or *List* object, a Select event is generated when the user selects an item from the list, whether by pressing the arrow keys or by clicking the left mouse button.

For a *MenuItem*, a Select event is generated when the user chooses the item.

For all other objects, this event is generated when the user presses the keys associated with the object's *Accelerator* property.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	Event code

4.120 SelectCertificate Event 848

Applies To: *HTMLRenderer*

Description

This event is triggered when *HTMLRenderer* requests a resource from a server that requires a certificate.

The event message reported as the result of `DBQ`, or supplied as the right argument to your callback function, is a 7-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'SelectCertificate' or 848
[3]	Index	Integer (see below)
[4]	Addr	Host address
[5]	Port	Host port
[6]		'is proxy'
[7]	Certificates	See below

Certificates is a vector of namespaces, each of which represents an available certificate and contains the following variables:

Name	Description
DER	Distinguished Encoding Rules. Character Vector.
Subject	Namespace (see below)
Issuer	Namespace (see below)
SerialNumber	Integer

The **Subject** and **Issuer** namespaces contain the following variables:

Name	Description
CommonName	Character vector
CountryName	Character vector
DisplayName	Character vector

The application should respond to this event by selecting a certificate from the list of available certificates reported by the 7th element of the event message. This is done by having a callback function that sets the 3rd element of the event message (Index) to the 0-origin index in Certificates and returns the event message as its result.

EXAMPLE

```

▼ arg←cb arg
[1]  a SelectCertificate callback function
[2]  arg[3]←0 a Select the first certificate
▼

```

4.121 ServiceNotification Event 94

Applies To: *Root*

Description

This event is generated in an APL service whenever the Windows Service Control Manager (SCM) requests a change of state. See *Installation/Configuration: APL Application As A Service*.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ServiceNotification' or 94
[3]	Action	Integer.
[4]	State	7-element integer vector

For further details, see the on-line documentation for `SERVICE_STATE` and the function `HashDefine` in the sample workspace `apl service`.

The state of a Windows service is determined by the user and the Windows Service Control Manager (SCM).

When the SCM requests a change of state, the APL interpreter responds by setting its state to the corresponding *pending* level (`SERVICE_STOPPED_PENDING`, `SERVICE_RUNNING_PENDING` or `SERVICE_PAUSED_PENDING`) and then generates a *ServiceNotification* event.

It is the responsibility of the APL service to process this event, perform the appropriate application tasks, and then respond (to the SCM) by calling the *SetServiceState* method to confirm that the service has reached the desired state.

4.122 SetColSize Event 176

Applies To: *Grid*, *ListView*

Description

If enabled, this event is reported when the user changes the width of a column in a *Grid* or *ListView* object, or changes the width of the row titles in a *Grid*. This may be done by dragging a border with the mouse or by double-clicking over a border. In the former case, the default action is to adjust the width of the appropriate column or the width of the row title area to the size selected by the user. In the latter case, the default action is to adjust the width to the maximum required to display all the data.

In either case, you can disable the default action by setting the event action code to `¬1` or you can selectively prevent a particular resize operation from taking place by returning 0 from a callback function.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'SetColSize' or 176
[3]	Column number	Integer. This is sensitive to the index origin, <code>⌈IO</code> , but is <code>¬1</code> if the user has resized the row titles in a <i>Grid</i> .
[4]	Width	Integer containing the value of the (new)column width. This is <code>¬3</code> if the user has double-clicked to request automatic width adjustment.
[5]	Undo flag	0 or 1

You can resize a column or resize the row titles under program control by calling *SetColSize* as a method. If you specify `¬1` as the *Width* parameter, the column will be resized to its default width .. If you specify a value of `¬2` the column will be resized to fit the data. The following expression will size the first *NCOLS* columns of a *Grid* called *F.G* to fit the data and the column titles:

```
{F.G.SetColSize ω ¬3}¨1:NCOLS
```

The Undo flag is applicable only to a Grid object and is always 1 if the event was generated by the user.

4.123 SetItemPosition Event 322

Applies To: *ListView*

Description

If enabled, this event is reported when the user drag-drops an item within a *ListView* object. This operation may be disabled by returning 0 from a callback function.

The event message reported as the result of `WM_DQ`, or supplied as the right argument to your callback function, is a 7-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'SetItemPosition' or 322
[3]	Item number	Integer. The index of the item.
[4]	Y-position	Integer. New y-position of the item.
[5]	X-position	Integer. New x-position of the item.
[6]	Button number	Integer. The mouse button used to perform the drag.
[7]	Shift State	Integer: Sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down 4 = Alt key is down

4.124 SetRowSize Event 175

Applies To: *Grid*

Description

If enabled, this event is reported when the user changes the height of a row or changes the height of the column titles. This may be done by dragging a border with the mouse or by double-clicking over a border. In the former case, the default action is to adjust the height of the appropriate row or the height of the column title area to the size selected by the user. In the latter case, the default action is to adjust the height to the maximum required to display all the data.

In either case, you can disable the default action by setting the event action code to `-1` or you can selectively prevent a particular resize operation from taking place by returning 0 from a callback function.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'SetRowSize' or 175
[3]	Row number	Integer. This is sensitive to the index origin, <code>⌈IO</code> , but is <code>-1</code> if the user has resized the column titles.
[4]	Height	Integer containing the value of the (new) row height. This is <code>-3</code> if the user has double-clicked to request automatic height adjustment.
[5]	Undo flag	0 or 1

You can resize a row or resize the column titles under program control by calling `SetRowSize` as a method. If you specify `-1` as the *Height* parameter, the row will be resized to its default height .. If you specify a value of `-2` the row will be resized to fit the data. The following expression will set the heights of first `NROWS` rows of a Grid called `F.G` to fit the data and the row titles.

```
{F.G.SetRowSize ⍵ -3}¨1NROWS
```

The Undo flag is always 1 if the event was generated by the user.

4.125 SetSpinnerText Event 421

Applies To: *Spinner*

Description

If enabled, this event is generated when the user clicks one of the spin buttons in a *Spinner* object. The event is reported *after* the value of the `Thumb` property has been updated but *before* the `Text` property has been changed. You may use this event to set the text in the *Spinner* dynamically instead of relying on it being updated automatically.

The event message reported as the result of `⌈DQ`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'SetSpinnerText' or 421
[3]	Thumb value	Integer. The new value of the Thumb property resulting from the user pressing one of the spin buttons.
[4]	Text	The text that is about to be put into the edit field.

The SetSpinnerText event is designed to allow you to dynamically set the text in the Spinner in response to a spin button. It might be used in circumstances where the set of items you wish to present to your user is not predictable in advance.

4.126 ShowComment Event 223

Applies To: *Grid*

Description

If enabled, a *Grid* will generate a ShowComment event when the user rests the mouse pointer over a commented cell. You may use this event to modify the appearance of the comment dynamically.

The event message reported as the result of `OnDQ` or supplied as the right argument to your callback function is an 8-element vector containing the following:

[1]	Object	ref or character vector
[2]	Event	'ShowComment' or 223
[3]	Cell row	integer
[4]	Cell column	integer
[5]	Comment text	character vector
[6]	Window height	integer, pixels
[7]	Window width	integer, pixels
[8]	Tip behaviour flag	(1 = yes; 0 = no)

A callback function may modify the standard behaviour. You may prevent the comment from being displayed by returning 0 as the result of the callback. Alternatively, you may modify the comment text, its window size, or its pop-up behaviour by changing the

appropriate element(s) of the event message and returning the new event message as the result.

Note that if the comment window relates to a row or column *title*, the value reported in element [3] or [4] of the event message is -1.

You may display the comment associated with a particular cell under program control by calling `ShowComment` as a method. In this case, only the *Cell row* and *Cell column* parameters need be specified. If however, you wish to override the comment text and/or its window size, you may do so (temporarily) by specifying the corresponding parameters. By default, a comment displayed under program control does not exhibit tip behaviour but remains visible until it is explicitly removed using the `HideComment` method.

Note that a comment will only be displayed if the specified cell is marked as a commented cell.

4.127 Spin Event 420

Applies To: *Spinner, UpDown*

Description

If enabled, this event is generated when the user clicks one of the spin buttons in a *Spinner* object. The event is reported *before* the value of the `Thumb` property has been updated. You may disable the operation of the spin buttons by disabling this event. You may selectively prevent the user selecting a particular value by returning 0 from a callback function. You may also return a modified event message as a result in order to set the `Thumb` property to a different value.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Spin' or 420
[3]	Thumb value	Integer. The new value of the <code>Thumb</code> property resulting from the user pressing one of the spin buttons.
[4]	Adjustment	Integer. The amount by which the new value of the <code>Thumb</code> differs from its previous value.

4.128 Splitting Event 281

Applies To: *Splitter*

Description

If enabled, this event is reported while a *Splitter* object is being dragged, between a *StartSplit* and an *EndSplit*. This event is only reported if full-drag is enabled.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to -1 or by returning 0 from a callback function.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Splitting' or 281
[3]	Y	y-position of top left corner
[4]	X	x-position of top left corner
[5]	H	height of the Splitter
[6]	W	width of the Splitter

See also *StartSplit*, *EndSplit*.

4.129 StartSplit Event 280

Applies To: *Splitter*

Description

If enabled, this event is reported when the user depresses the left mouse button over a *Splitter* object to signify the beginning of a drag operation.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to -1 or by returning 0 from a callback function.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'StartSplit' or 280

See also *EndSplit*, *Splitting*.

4.130 StateChange Event 35

Applies To: *Form*, *SubForm*

Description

This event is generated by a *Form* or *SubForm* when the user attempts to change the *State* of a *Form*, by minimising it, maximising it, or restoring it from a minimised or maximised state. The event is reported **before** the window changes state. You may prevent the state change by disabling the event (action code `-1`) or by returning a 0 result from an attached callback function.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'StateChange' or 35
[3]	Window state	0 (about to be restored) 1 (about to be minimised) 2 (about to be maximised)

4.131 SysColorChange Event 134

Applies To: *Root*

Description

If enabled, this event is reported when the user or another application updates the system colour palette. The event is reported after the change has taken place and cannot be disabled or inhibited in any way. If you want your application to respond to colour palette changes, this event gives you the opportunity of doing so.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'SysCoLoRChange' or 134

4.132 TCPAccept Event 371

Applies To: *TCPSocket*

Description

If enabled, this event is reported when a client connects to a server *TCPSocket* object.

You may not disable or nullify the operation by setting the action code for the event to `-1` or by returning 0 from a callback function. You may also not call *TCPAccept* as a method or generate this event artificially using `□NQ`.

The event message reported as the result of `□DQ`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'TCPAccept' or 371
[3]	Socket handle	an integer

The socket handle reported by this event is the socket handle for the original listening socket that was associated with the *TCPSocket* before the client connected.

If you want your server to remain available for other clients, you must create a new *TCPSocket* object in a callback function attached to this event. The new *TCPSocket* object must be created by cloning the original listening socket. This is done by specifying the socket handle as the value of its `SocketNumber` property. You may not specify any other properties (except `Event` and `Data`) in the `□WC` statement that creates the new clone object.

The default processing for this event is to close the socket handle reported by the 3rd element of the event message **unless** it has been associated with a new *TCPSocket* object by the callback function as described above. You may prevent this from occurring by returning 0 from a callback function. This may be necessary in a multithreaded application.

You may not call *TCPClose* as a method or generate this event artificially using `□NQ`.

4.133 TCPClose Event 374

Applies To: *TCPSocket*

Description

If enabled, this event is reported when the remote end of a TCP/IP connection breaks the connection.

You may not disable or nullify the operation by setting the action code for the event to `-1` or by returning 0 from a callback function. You may also not call `TCPClose` as a method or generate this event artificially using `□NQ`.

The event message reported as the result of `□DQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' TCPClose ' or 374

4.134 TCPConnect Event 372

Applies To: *TCPSocket*

Description

If enabled, this event is reported when a server accepts the connection of a client *TCPSocket* object and is reported by the client.

You may not disable or nullify the operation by setting the action code for the event to `-1` or by returning 0 from a callback function. You may also not call `TCPConnect` as a method or generate this event artificially using `□NQ`.

The event message reported as the result of `□DQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' TCPConnect ' or 372

4.135 TCPErrors Event 370

Applies To: *TCPSocket*

Description

This event is generated when a fatal TCP/IP error occurs and is reported by a *TCPSocket* object.

The default processing for this event is to display a message box containing details of the TCP/IP error. You may disable the display of this message box by setting the action code for the event to -1 or by returning 0 from a callback function attached to it.

The event message reported as the result of `GetEvent`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'TCPErrors' or 370
[3]	Error code	a number
[4]	Error text	a character vector

4.136 TCPGotAddr Event 377

Applies To: *TCPSocket*

Description

If enabled, this event is reported when a host name (specified by the *RemoteAddrName* or *LocalAddrName* property) is resolved to an IP address.

You may not disable or nullify the operation by setting the action code for the event to -1 or by returning 0 from a callback function. You may also not call `TCPGotAddr` as a method or generate this event artificially using `SendMessage`.

The event message reported as the result of `GetEvent`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'TCPGotAddr' or 377

Note that the IP address is not reported in the event message but may be obtained from *RemoteAddr* or *LocalAddr* as appropriate.

4.137 TCPGotPort Event 378

Applies To: *TCPSocket*

Description

If enabled, this event is reported when a port name (specified by the *RemotePortName* or *LocalPortName* property) is resolved to a port number.

You may not disable or nullify the operation by setting the action code for the event to -1 or by returning 0 from a callback function. You may also not call *TCPGotPort* as a method or generate this event artificially using `□NQ`.

The event message reported as the result of `□DQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' TCPGotPort ' or 378

Note that the port number is not reported in the event message but may be obtained from *RemotePort* or *LocalPort* as appropriate.

4.138 TCPReady Event 379

Applies To: *TCPSocket*

Description

If enabled, this event is reported when the TCP/IP buffers are free and there is no data waiting to be sent in the internal APL queue.

This event is provided to enable you to control the transmission of a large amount of data that cannot be handled in a single call to *TCPSend*.

The amount of data that the system can handle in one go is limited by TCP/IP buffers, the speed of the network, and the amount of Windows memory and disk space available for buffering.

You may not disable or nullify the operation by setting the action code for the event to `-1` or by returning `0` from a callback function. However, you *may* call `TCPReady` as a method or generate this event artificially using `□NQ`.

The event message reported as the result of `□DQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'TCPReady' or 379

4.139 TCPRecv Event 373

Applies To: *TCPSocket*

Description

If enabled, this event is reported when data is received by a *TCPSocket* object.

You may not disable or nullify the operation by setting the action code for the event to `-1` or by returning `0` from a callback function. You may also not call `TCPRecv` as a method or generate this event artificially using `□NQ`.

The event message reported as the result of `□DQ`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'TCPRecv' or 373
[3]	Data	the data received
[4]	IP address	character vector
[5]	Port number	integer

Elements [4-5] refer to the IP address and port number of the remote process that sent the data.

If the *SocketType* is 'Stream', this information will be identical to the values of the *RemoteAddr* and *RemotePort* respectively.

If the *SocketType* is 'UDP' and there is potentially more than one partner sending you data, the IP address and port number information provided by the `TCPRecv` event is

more reliable than the current values of *RemoteAddr* and *RemotePort* as these may already have changed.

4.140 ThumbDrag Event 440

Applies To: *Scroll*, *TrackBar*

Description

If enabled, this event is generated when the user drags the thumb in a *TrackBar* object. The event is reported *after* the value of the Thumb property has been updated and is reported continuously as the thumb is dragged. You may not disable this event or alter its effect with a callback function.

The event message reported as the result of `DQ`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ThumbDrag' or 440
[3]	Thumb value	Integer. The new value of the Thumb property resulting from the user dragging the thumb.

4.141 Timer Event 140

Applies To: *Timer*

Description

This event is generated at regular intervals by a *Timer* object and is typically used to fire a callback function to perform a task repeatedly. Returning a 0 from a callback function attached to a Timer event has no effect. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2 element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Timer' or 140

4.142 VScroll Event 38

Applies To: *Form, SubForm*

Description

If enabled, this event is generated when the user attempts to move the thumb in a vertical scrollbar in a *Form* or *SubForm*. This event occurs only in a *Form* whose *VScroll* property is set to `-1` and is distinct from the *Scroll* event which is generated by a *Scroll* object. The event may be generated in one of three ways :

1. dragging the thumb
2. clicking in one of the "arrow" buttons situated at the ends of the scrollbar. This is termed a small change, the size of which is defined by *Step[1]*.
3. clicking in the body of the scrollbar. This is termed a large change, the size of which is defined by *Step[2]*.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 4-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'VScroll' or 38
[3]	Scroll Type	numeric
[4]	Position	numeric

The value of Scroll Type is 0 (drag), 1 or `-1` (small change) or 2 or `-2` (large change). The sign indicates the direction.

The value of Position is the new (requested) position of the thumb. Notice however that the event is generated **before** the thumb is actually moved. If your callback function returns a scalar 0, the position of the thumb will remain unaltered.

4.143 VThumbDrag Event 441

Applies To: *Form, SubForm*

Description

If enabled, this event is generated when the user attempts to drag the thumb in a vertical scrollbar in a *Form* or *SubForm*. This event occurs only in a *Form* or *SubForm*

whose *HScroll* property is set to `-1` and is distinct from the *Scroll* event that is generated by a *Scroll* object.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'VThumbDrag' or 441
[3]	Position	numeric

The value of *Position* is the new (requested) position of the Thumb. Setting the action code of this event to `-1`, or returning a 0 from a callback function attached to it, has no effect.

4.144 WebSocketError Event 844

Applies To: *HTMLRenderer*

Description

This event is triggered an error occurs on the WebSocket. It is for notification only.

The event message reported as the result of `⌈DQ`, or supplied as the right argument to your callback function, is a 4-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'WebSocketError' or 844
[3]	ID	Character vector containing the ID of the WebSocket
[4]	Message	The error message (character vector)

4.145 WebSocketReceive Event 842

Applies To: *HTMLRenderer*

Description

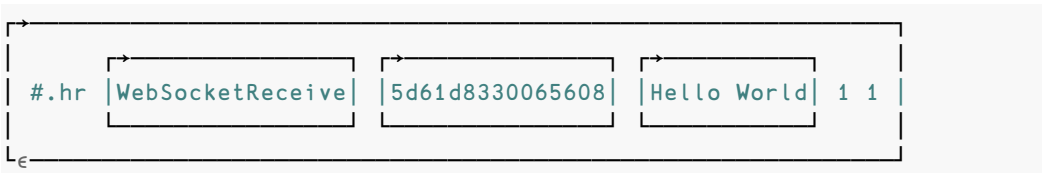
This event is triggered when data is received over a WebSocket. This event is reported for information only. The result (if any) of a callback function will be ignored.

Note that the WebSocket protocol provides for the possibility for the data to be sent in chunks, causing a succession of `WebSocketReceive` events. The FIN bit of the last chunk will be 1. The CEF does not currently implement "chunking", so FIN will always be 1.

The event message reported as the result of `OnWebSocketReceive`, or supplied as the right argument to your callback function, is a 6-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'WebSocketReceive' or 842
[3]	ID	Character vector containing the ID of the WebSocket
[4]	Data	Character or integer vector.
[5]	FIN	Boolean. 1 indicates that this is the last chunk; 0 that there is more to come.
[6]	Datatype	1 = character, 2 - numeric values in the range -128 to 127.

EXAMPLE



4.146 WebSocketUpgrade Event 841

Applies To: *HTMLRenderer*

Description

This event is reported when the client component of an *HTMLRenderer* object opens a WebSocket and the requested URL matches a pattern specified by the *InterceptedURLs* property. If there is no match, the connection request is processed as an external request by the [Chromium Embedded Framework \(CEF\)](#).

The event message reported as the result of `OnWebSocketUpgrade`, or supplied as the right argument to your callback function, is a 6-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'WebSocketUpgrade' or 841
[3]	ID	Character vector containing the ID of the WebSocket
[4]	URL	The requested URL of the WebSocket
[5]	Headers	ASCII including CRLF
[6]	Type	Character vector 'auto' or 'manual'

The protocol for establishing the connection is defined by *InterceptedURLs* and is reported by the 6th element (Type) of the event message.

If Type is 'auto', the protocol is handled internally and this event is reported when the connection has already been made. Should the connection fail, a *WebSocketError* event will be reported instead.

If Type is 'manual', a callback function for *WebSocketUpgrade* is mandatory and is responsible for completing (or denying) the connection. This is achieved by setting the 5th element of the event message (Headers) to indicate an appropriate positive or negative response to the request and returning the entire event message as its result. If a valid response is not generated in this way, the connection will time-out causing a *WebSocketError* event. For further information, see <https://tools.ietf.org/html/rfc6455#section-1.3>.

In both cases, the WebSocket ID is subsequently required to send a message by calling the *WebSocketSend* method or to close the connection using the *WebSocketClose* method.

Note that several WebSocket connections may be made concurrently.

EXAMPLE



4.147 WinIniChange Event 133

Applies To: Root

Description

If enabled, this event is reported when another application changes relevant registry settings using the standard API calls. The event is reported after the change has taken place and cannot be disabled or inhibited in any way. If your application depends upon registry settings, this event gives you the opportunity of refreshing these parameters if they are changed.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'WinIniChange' or 133

5 Methods

5.1 Abort Method 103

Applies To: *Printer*

Description

This method causes the print job to be aborted and all pending output to be discarded.

The Abort method is niladic.

If you attach a callback function to this method and have it return a value of 0, the print job will continue.

5.2 AddChildren Method 310

Applies To: *TreeView*

Description

This method is used to add child items to an item in a *TreeView* object

The argument to AddChildren is a 3, 4 or 5 element array as follows:

[1]	Item number	Integer.
[2]	New items	Vector of character vectors.
[3]	Depth vector	Integer vector.
[4]	Picture vector	Integer vector.
[5]	Selected picture vector	Integer vector.

Item number specifies the index of the item to which the child items are to be added.

New items is a vector of character vectors containing the labels for the new child items.

Depth vector is an integer vector specifying the depth of each of the new items relative to the parent item to which they are being added. The first element of this array must be 0.

Picture vector and *Selected picture vector* are optional and specify values of *ImageIndex* and *SellImageIndex* respectively for each of the new items.

The result is the index at which the first new item has been inserted.

5.3 AddComment Method 220

Applies To: *Grid*

Description

This method is used to add a new comment to a *Grid*.

The argument to AddComment is a 3, 4 or 5 element array as follows:

[1]	Row	integer
[2]	Column	integer
[3]	Comment text	character array
[4]	Height in pixels	integer
[5]	Width in pixels	integer

For example, the following statement associates a comment with the cell at row 2, column 1; the text of the comment is "Hello", and the size of the comment window is 50 pixels (high) by 60 pixels (wide).

```
F.G.AddComment 2 1 'Hello' 50 60
```

Note that if you specify a row number of -1, the comment is added to the corresponding column *title*. Similarly, if you specify a column number of -1, the comment is added to the corresponding row *title*.

The height and width of the comment window, specified by the last 2 elements of the argument are both optional. If the cell already has an associated comment, the new comment replaces it.

You can use a dfn to add several comments in one statement; for example:

```
(1 2)(2 3){F.G.AddComment α,ω}''Hello' 'Goodbye'
```

Note that just before the comment is displayed, the *Grid* generates a *ShowComment* event which gives you the opportunity to (temporarily) change the text and/or window size of a comment dynamically.

The comment text specified by the 5th element of the argument to *Dyalog APL Language*: *Nq* must be a simple character scalar, vector, matrix or vector of vectors. Text specified by a simple character vector will be wrapped automatically if necessary. A matrix or vector of vectors may be used to explicitly specify multi-line text. If the array is a vector whose first element is an opening brace {}, the text is assumed to be in rich-text format (RTF) and is displayed accordingly. Note that there is no way for the user to scroll the text in the comment window and it is entirely your responsibility to ensure that the size of the window is appropriate for its contents.

5.4 AddItems Method 308

Applies To: *TreeView*

Description

This method is used to add items to a *TreeView* object

The argument to *AddItems* is a 3, 4 or 5-element array as follows:

[1]	Item number	Integer.
[2]	New items	Vector of character vectors.
[3]	Depth vector	Integer vector.
[4]	Picture vector	Integer vector.
[5]	Selected picture vector	Integer vector.

Item number specifies the index of the item to which the child items are to be added.

New items is a vector of character vectors containing the labels for the new child items.

Depth vector is an integer vector specifying the depth of each of the new items relative to the parent item to which they are being added. The first element of this array must be 0. This element may be omitted. If so, it is assumed to be all 0s.

Picture vector and *Selected picture vector* are optional and specify values of *ImageIndex* and *SellImageIndex* respectively for each of the new items.

The new items are inserted with the first one being placed at the same level in the hierarchy as the item specified in element [1].

The result is an integer that reports the index position at which the first of the new items has been inserted.

5.5 AnimClose Method 291

Applies To: *Animation*

Description

The AnimClose method closes the AVI file that is currently loaded in an Animation object. The display is reset to the object's background colour.

The AnimClose method is niladic.

5.6 AnimOpen Method 290

Applies To: *Animation*

Description

The AnimOpen method opens an AVI file in an Animation object.

The argument to AnimOpen is a 1 or 2-element array as follows:

[1]	File	character vector
[2]	Resource id	integer

If a single element is specified, it represents the name of a .AVI file.

If 2 elements are specified, the first element specifies the name of a DLL or EXE and the second element identifies the particular AVI resource stored in that file. The identifier may be its name (a character string) or its resource id (a non-zero positive integer).

If the *AutoPlay* property is set to 1, the animation will play immediately. Otherwise, only the first frame will be displayed.

Note that the Animation object can only play AVI files or resources that have no sound and can only display uncompressed AVI files or .AVI files that have been compressed using Run-Length Encoding (RLE). If you attempt to open an inappropriate AVI file, the

operation will fail with a DOMAIN ERROR and the following message will be displayed in the Status Window:

AVI file includes sound data or is in a format not supported by the Animation object

5.7 AnimPlay Method 292

Applies To: *Animation*

Description

The AnimPlay method plays an AVI clip in an Animation object.

The argument to AnimPlay is a 3-element array as follows:

[1]	Repeat	integer
[2]	From	integer
[3]	To	integer

Repeat specifies the number of times the clip is repeated. A value of -1 causes the clip to be repeated indefinitely.

From is a 0-based index of the frame where playing begins and must be less than 65536. A value of zero means begin with the first frame in the AVI clip

To is a 0-based index of the frame where playing ends and must be less than 65536. A value of -1 means end with the last frame in the AVI clip

The last frame remains displayed until the clip is unloaded using *AnimClose* or until another clip is loaded.

5.8 AnimStop Method 293

Applies To: *Animation*

Description

The AnimStop method stops playing an AVI clip in an Animation object.

AnimStop is niladic.

The last frame remains displayed until the clip is unloaded using *AnimClose* or until another clip is loaded.

5.9 Animate Method 29

Applies To

Description

The Animate method enables you to produce special effects when showing or hiding objects. There are three types of animation: roll, slide, and alpha-blended fade.

The argument to Animate is a 1 or 2-element array as follows:

[1]	Effects	integer
[2]	Play time	integer

The value of the *Effects* parameter is the sum of the following flags:

Flag	Value	Description
AW_HOR_POSITIVE	1	Animates the window from left to right. This flag can be used with roll or slide animation. It is ignored when used with the AW_CENTER flag.
AW_HOR_NEGATIVE	2	Animates the window from right to left. This flag can be used with roll or slide animation. It is ignored when used with the AW_CENTER flag.
AW_VER_POSITIVE	4	Animates the window from top to bottom. This flag can be used with roll or slide animation. It is ignored when used with the AW_CENTER flag.
AW_VER_NEGATIVE	8	Animates the window from bottom to top. This flag can be used with roll or slide animation. It is ignored when used with the AW_CENTER flag.
AW_CENTER	16	Makes the window appear to collapse inward if being hidden or expand outward if being displayed
AW_SLIDE	262144	Uses slide animation. By default, roll animation is used. This flag is meaningless on its own but is ignored when used with the AW_CENTER flag.
AW_BLEND	524288	Uses a fade effect. This flag can be used only for a Form.

The Playtime parameter is optional and specifies the length of time over which the animation is played in milliseconds. The default value depends upon the animation but is typically 200 milliseconds.

5.10 Browse Method 585

Applies To: *OCXClass*, *OLEClient*

Description

The Browse method is retained for compatibility with previous versions of Dyalog APL, but is no longer relevant and has no effect.

5.11 CancelToClose Method 367

Applies To: *PropertySheet*

Description

This method is used to change the buttons in a *PropertySheet* object. Its effect is to disable the Cancel button and, if the Style of the *PropertySheet* is 'Standard', it changes the text of the OK button to "Close". There is no result.

The CancelToClose method is niladic.

5.12 CellFromPoint Method 200

Applies To: *Grid*

Description

This method converts from Grid co-ordinates to cell co-ordinates.

The argument to CellFromPoint is a 2-element array as follows:

[1]	y-coordinate	number in Grid co-ordinates
[2]	x-coordinate	number in Grid co-ordinates

The result is a 2-element vector containing the following:

[1]	y-coordinate	number in cell co-ordinates
[2]	x-coordinate	number in cell co-ordinates

5.13 ChooseFont Method 240

Applies To

Description

This method is used to display the standard Windows font selection dialog box.

The argument to ChooseFont is $\emptyset$ or a 1 or 2-element array as follows:

[1]	Printer name	character scalar or vector.
[2]	Modify flag	0 or 1.

If the argument is $\emptyset$ or the first element of the argument is ' ', the user is offered a list of fonts suitable for use on the screen. If not, the user is offered a choice of fonts suitable for the specified *Printer* object. If you omit the 2nd element, the modify flag defaults to 0.

The dialog box is initialised with the properties of the *Font* object specified in the first element of the event message.

When the user presses the "OK" button, the "Cancel" button or closes the dialog box, ChooseFont terminates. Its result is either 0 (user pressed "Cancel") or a 2-element vector. In the latter case, the first element is an 8-element array that describes the selected font as described below, and the second element is a 3-element RGB colour vector.

[1]	Face name of selected font (character vector)
[2]	Character height in pixels (integer)
[3]	Fixed width or not (Boolean)
[4]	Italic or not (Boolean)
[5]	Underline or not (Boolean)
[6]	Weight (integer)
[7]	Angle of rotation (integer)
[8]	Character set (see <i>CharSet</i>)

If the modify flag was 1, the *Font* object is redefined to match the user's selections and all the objects that reference the *Font* are redrawn.

5.14 ColChange Method 159

Applies To: *Grid*

Description

This method is used to change the data in a column of a *Grid*.

The argument to ColChange is a 2-element array as follows:

[1]	Column number	integer
[2]	Column data	array

Note that the *Column data* must be a scalar or a vector whose length is equal to the number of rows in the *Grid*. Its elements may be scalar numbers, character vectors or matrices.

5.15 ColSorted Method 174

Applies To: *Grid*

Description

This method is used to specify that an image is to be displayed in a *Grid* column title to indicate the column has been sorted.

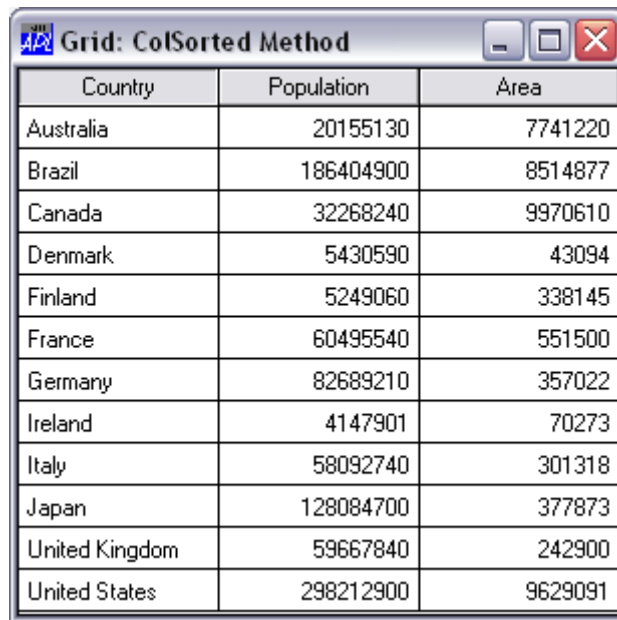
The argument to ColSorted is a 2-element array as follows:

[1]	Column number	Integer.
[2]	Sorted state	Integer. -1 = Sorted Down 0 = Unsorted 1 = Sorted Up

The column title for the appropriate column is redrawn to include the appropriate image.

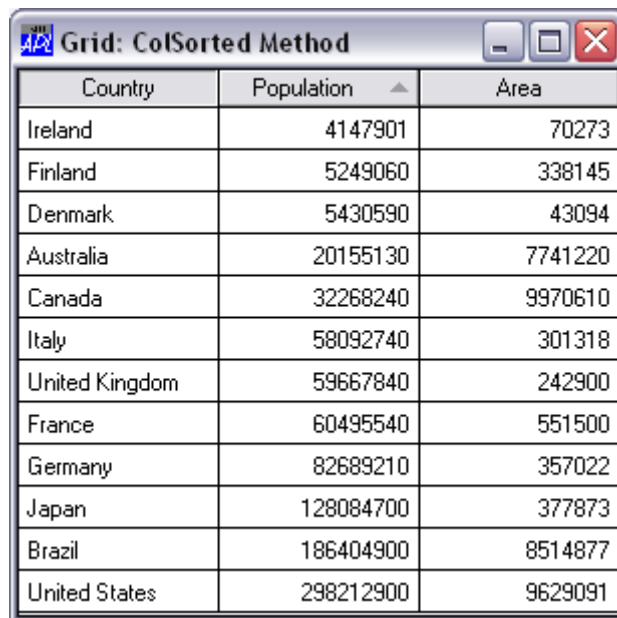
If you wish to use your own images, you may specify them using the *ColSortImages* property.

```
'F'[]WC'Form' 'Grid: ColSorted Method'
'F.G'[]WC'Grid'('Posn' 0 0)(100 100)
F.G.Values←(COUNTRIES,POPULATION,[1.5]AREA)
F.G.ColTitles←'Country' 'Population' 'Area'
F.G.TitleWidth←0
```



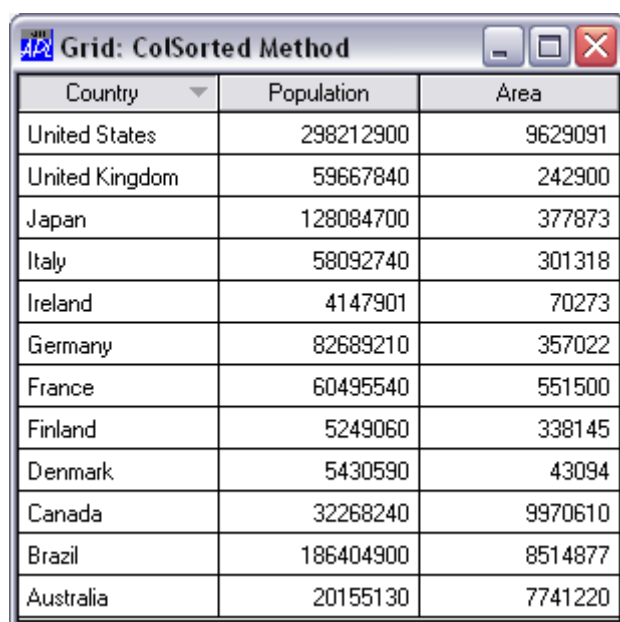
Country	Population	Area
Australia	20155130	7741220
Brazil	186404900	8514877
Canada	32268240	9970610
Denmark	5430590	43094
Finland	5249060	338145
France	60495540	551500
Germany	82689210	357022
Ireland	4147901	70273
Italy	58092740	301318
Japan	128084700	377873
United Kingdom	59667840	242900
United States	298212900	9629091

```
F.G.Values←(Values[⌵Values[;2];])
F.G.ColSorted 2 1
```



Country	Population ▲	Area
Ireland	4147901	70273
Finland	5249060	338145
Denmark	5430590	43094
Australia	20155130	7741220
Canada	32268240	9970610
Italy	58092740	301318
United Kingdom	59667840	242900
France	60495540	551500
Germany	82689210	357022
Japan	128084700	377873
Brazil	186404900	8514877
United States	298212900	9629091

```
F.G.(Values←Values[⌶↑Values[;1];])
F.G.ColSorted 2 0
F.G.ColSorted 1 ⌶1
```



Country	Population	Area
United States	298212900	9629091
United Kingdom	59667840	242900
Japan	128084700	377873
Italy	58092740	301318
Ireland	4147901	70273
Germany	82689210	357022
France	60495540	551500
Finland	5249060	338145
Denmark	5430590	43094
Canada	32268240	9970610
Brazil	186404900	8514877
Australia	20155130	7741220

5.16 DateToIDN Method 264

Applies To: *Calendar*, *DateTimePicker*, *Root*

Description

This method is used to convert a date from $\square$ TS format into an *IDN* suitable for use in a *Calendar* object.

The argument to *DateToIDN* is a 3-element array as follows:

[1]	Year	Integer
[2]	Month	Integer
[3]	Day	Integer

DateToIDN will also accept a single enclosed argument containing these values. In either case, if you specify more than 3 numbers, excess elements they will be ignored.

EXAMPLES

```
F.C.DateToIDN 1998 9 11
36048
F.C.DateToIDN <1998 9 11
36048
F.C.DateToIDN []TS
36048
F.C.DateToIDN,[]TS
36048
```

5.17 DelCol Method 155

Applies To: *Grid*

Description

This method is used to delete a specified column from a *Grid* object.

The argument to DelCol is a 1 or 2-element vector as follows:

[1]	Column number	number of the column (integer) to delete
[2]	Undo flag	0 or 1 (optional; default 0)

If the *Undo flag* 1, the column may subsequently be restored by invoking the *Undo* method. If the *Undo flag* is omitted or is 0, the operation may not be undone.

5.18 DelComment Method 221

Applies To: *Grid*

Description

This method is used to delete a comment from a *Grid*.

The argument to DelComment is a 2 array as follows or $\emptyset$:

[1]	Row	integer
[2]	Column	integer

For example, the following expression removes the comment associated with the cell at row 2, column 1.

```
F.C.DelComment 2 1
```

Note that to delete a comment associated with a row or column *title*, the appropriate element in the argument should be `-1`.

If the argument is `0`, all comments are deleted.

5.19 DelRow Method 154

Applies To: *Grid*

Description

This method is used to delete a specified row from a *Grid* object.

The argument to `DelRow` is a 1 or 2-element array as follows:

[1]	Row number	number of the row (integer) to delete
[2]	Undo flag	0 or 1 (optional; default 0)

If the *Undo flag* is 1, the column may subsequently be restored by invoking the *Undo* method. If the *Undo flag* is omitted or is 0, the operation may not be undone.

5.20 DeleteChildren Method 311

Applies To: *TreeView*

Description

This method is used to delete child items from a parent item in a *TreeView* object.

The argument to `DeleteChildren` is a scalar or 1 element array as follows:

[1]	Item number	Integer.

Item number specifies the index of the parent item from which the child items are to be removed.

The result is an integer that indicates the number of children that have been removed from the parent item.

5.21 DeleteItems Method 309

Applies To: *TreeView*

Description

This method is used to delete items from a *TreeView* object.

The argument to *DeleteItems* is a 2-element array as follows:

[1]	Item number	Integer.
[2]	Number of Items	Integer.

Item number specifies the index of the first item to be removed.

Number of items specifies the number of items to be removed and refers to those items *at the same level* in the *TreeView* hierarchy as the *Item number*. *Number of items* is optional and defaults to 1.

Note that any children of these items will also be removed.

The result is an integer that indicates the total number of items, including children, that have been removed from the *TreeView*.

5.22 DeleteTypeLib Method 521

Applies To: *Root*

Description

The *DeleteTypeLib* method removes a loaded Type Library from the workspace.

The argument to *DeleteTypeLib* is as follows:

[1]	TypeLib	character vector

The Type Library may be identified by its name or by its class id.

The result is 0, 1 or -1.

If successful, the specified Type Library, and all dependent Type Libraries not referenced by any other currently loaded Type Libraries, are removed from the active workspace. The result is 1.

If the specified Type Library is in use, no action is taken and the result is 0.

If the argument is not the name or CLSID of a loaded Type Library, no action is taken and the result is -1.

5.23 Detach Method 270

Applies To

Description

This method is used to detach the GUI component from an object without losing the functions, variables and sub-namespaces that it may contain.

The Detach method is niladic.

The effect of this method is to remove the GUI component associated with the named object, leaving behind a plain namespace of the same name. All non-GUI child objects are retained. GUI child objects are either destroyed, or similarly converted to plain namespaces depending upon the values of their KeepOnClose properties.

5.24 DuplicateColumn Method 178

Applies To: Grid

Description

This method is used to duplicate a column in a *Grid* object.

The argument to DuplicateColumn is a 2, 3, 4 or 5-element vector as follows:

[1]	Source Column number	number of the column (integer) to be duplicated
[2]	Target Column number	new column number (integer)
[3]	Comment flag	0 or 1 (optional, default 1)
[4]	Lock flag	0 or 1 (optional, default 1)
[5]	Undo flag	0 or 1 (optional; default 0)

If the *Comment flag* is 1 (the default), any Comments associated with cells in the source column are duplicated in the target column.

If the *Lock flag* is 1 (the default), the lock state of the column is duplicated; otherwise, the new column is not locked.

If the *Undo flag* is 1, the column may subsequently be restored by invoking the Undo method. If this element is omitted or is 0, the operation may not be undone.

5.25 DuplicateRow Method 177

Applies To: *Grid*

Description

This method is used to duplicate a row in a *Grid* object.

The argument to DuplicateRow is a 2, 3, 4 or 5-element vector as follows:

[1]	Source Row number	number of the row (integer) to be duplicated
[2]	Target Row number	new row number (integer)
[3]	Comment flag	0 or 1 (optional, default 1)
[4]	Lock flag	0 or 1 (optional, default 1)
[5]	Undo flag	0 or 1 (optional; default 0)

If the *Comment flag* is 1 (the default), any Comments associated with cells in the source row are duplicated in the target row.

If the *Lock flag* is 1 (the default), the lock state of the row is duplicated; otherwise, the new row is not locked.

If the *Undo flag* is 1, the row may subsequently be restored by invoking the Undo method. If this element is omitted or is 0, the operation may not be undone.

5.26 ExecuteJavaScript Method 839

Applies To: *HTMLRenderer*

Description

This method is used to execute JavaScript in an *HTMLRenderer* object.

The argument to `ExecuteJavaScript` is a single item as follows:

[1]	Code	character vector containing JavaScript code

The shy result of `ExecuteJavaScript` is currently 1; this may change.

EXAMPLE

```
hr.ExecuteJavaScript 'alert("Hello")'
```

5.27 FileRead Method 90

Applies To: *Bitmap, Cursor, Icon, Metafile, RichEdit*

Description

This method causes the object to be recreated from the file named in its *File* property.

The `FileRead` method is niladic.

[1]	Object	ref or character vector
[2]	Event	'FileRead' or 90

If you attach a callback function to this event and have it return a value of 0, the object will not be recreated from file.

5.28 FileWrite Method 91

Applies To: *Bitmap, Cursor, Icon, Metafile, RichEdit*

Description

This method causes the object to be written to the file named in its *File* property. For the *Bitmap* and *Icon* objects this method will write a file of type *.BMP* and *.ICO* to a file with the appropriate extension. If *File* specifies any other extension, the method will fail with a `DOMAIN ERROR`:

```
DOMAIN ERROR: This object cannot be saved to this type of file.
```

The `FileWrite` method is niladic.

If you attach a callback function to this event and have it return a value of 0, the object will not be written to file. You could use this to avoid overwriting an existing file.

5.29 Flush Method 135

Applies To: *Root*

Description

This method forces any objects that have been created but not yet shown to be displayed. Normally, Dyalog APL/W buffers the display of new objects unless they are being created by a callback function. This event can be used to override the buffering.

The Flush method is niladic.

5.30 GetBuildID Method 992

Applies To: *Root*

Description

This method is used to obtain the Build ID of a Dyalog executable or the checksum of a file.

The argument to GetBuildID is $\emptyset$ or a single item as follows:

[1]	File name	character vector

The (shy) result is an 8-element character vector of hexadecimal digits that represents the Build ID.

If the argument is $\emptyset$, the build id is that of the current version of Dyalog that is running the expression.

Note that although this method is designed to uniquely identify different versions of Dyalog by its checksum, it may be used to obtain a checksum for *any* arbitrary file.

EXAMPLES

```

    GetBuildID 0
38091b76
    GetBuildID 'E:\DYALOG81\DYALOG.EXE'
cbf0d376
    GetBuildID 'myfile'
4a29334d

```

Note that if the file does not exist, the result is 00000000.

5.31 GetCellRect Method 201

Applies To: *Grid*

Description

This method returns the rectangle associated with a particular cell in a Grid.

The argument to GetCellRect is a 2-element vector as follows:

[1]	Row	integer
[2]	Column	integer

The result is a 2-element nested vector. The first element contains the y and x-coordinate of the top-left corner of the cell. The second element contains the height and width of the cell.

The result is reported in terms of the coordinate system of the Grid object.

5.32 GetCommandLine Method 145

Applies To: *Root*

Description

The GetCommandLine method returns the command line that was used to start the current Dyalog APL session or application.

The GetCommandLine method is niladic.

The result is a character vector.

EXAMPLES

```
GetCommandLine
"C:\Program Files\Dyalog\Dyalog APL-64 13.2 Unicode\dyalog.exe"
```

```
⎕←2 ⎕NQ '.' 'GetCommandLine'
"C:\Program Files\Dyalog\Dyalog APL-64 13.2 Unicode\dyalog.exe"
```

Note

GetCommandLine only works on Windows, and its use is deprecated in favour of *GetCommandLineArgs*, which works on all platforms.

5.33 GetCommandLineArgs Method 148

Applies To: *Root*

Description

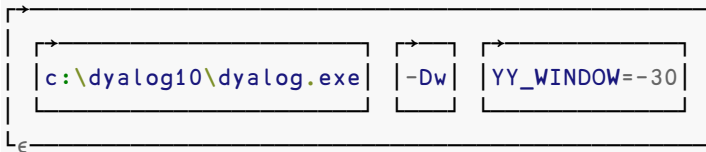
The GetCommandLineArgs method returns the command and the arguments to the command that was used to start the current Dyalog APL session or application.

The GetCommandLineArgs method is niladic.

The result is a vector of character vectors. For example:

```
GetCommandLineArgs
C:\Dyalog10\dyalog.exe -Dw YY_WINDOW=-30
```

```
DISPLAY 2 ⎕NQ '.' 'GetCommandLineArgs'
```



5.34 GetComment Method 222

Applies To: *Grid*

Description

This method is used to retrieve the comment associated with a cell in a *Grid*.

The argument to `GetComment` is a 2-element array as follows:

[1]	Row	integer
[2]	Column	integer

For example, the following expression retrieves the comment associated with the cell at row 3, column 1.

```
F.C.GetComment 3 1
1 3 Hello 175 100
```

Note that to retrieve a comment associated with a row or column *title*, the appropriate element in the argument should be `~1`.

If there is no comment associated with the specified cell, the result is a scalar 1.

5.35 GetEnvironment Method 510

Applies To: *Root*

Description

This method is used to obtain information about one or more parameters that were specified in the Dyalog command line, your Windows registry, or defined as environment variables. If a value is defined in several places (for example, MAXWS in the command line overriding MAXWS in the registry), `GetEnvironment` follows exactly the same logic as is used by Dyalog APL itself and so obtains the same value.

The argument to `GetEnvironment` is a single item as follows:

[1]	Parameter name(s)	see below

Parameter names is simple character vector or vector of character vectors specifying one or more parameters.

The result is a simple character vector or a vector of character vectors.

EXAMPLES

```
GetEnvironment 'MAXWS'
2G

GetEnvironment <'LOG_SIZE' 'MAXWS'
16 2G
```

Note that you may use GetEnvironment to obtain the values of your own arbitrary parameters given on the APL command line or specified as environment variables.

GetEnvironment is not supported by DYALOG.DLL because it does not use parameters.

5.36 GetEventInfo Method 551

Applies To: OCXClass, OLEClient

Description

This method is used to obtain information about a particular event or set of events supported by a COM object.

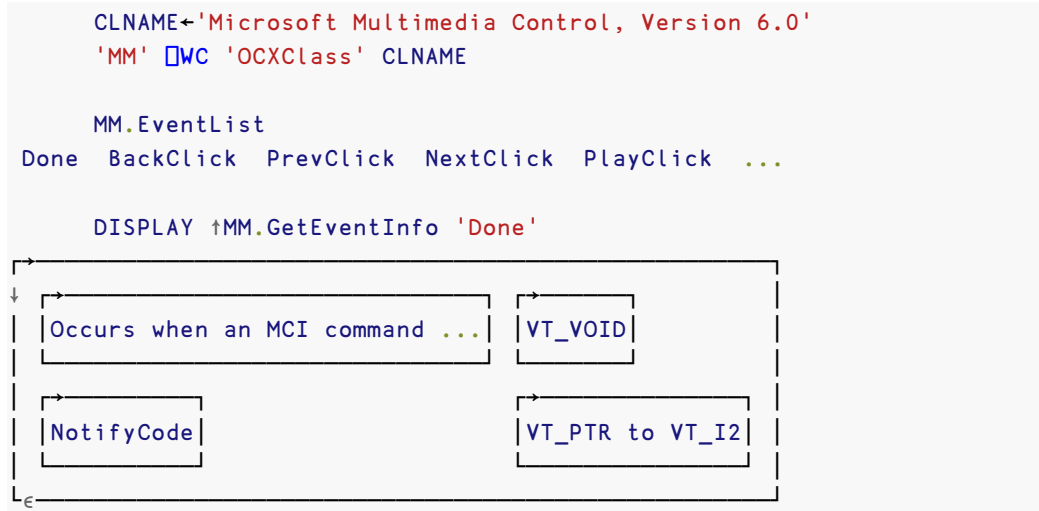
For each event supported by a COM object, the author will have registered the data type of its result (if it has a result), a help message or description of the event (optional) and the name and data type of each of its parameters. These event parameters make up the array returned by □□□ or supplied as an argument to your callback function. The GetEventInfo method returns this information.

The argument to GetEventInfo is a single item as follows:

[1]	Event name(s)	see below

Event name(s) is a simple character vector or a vector of character vectors specifying one or more names of events supported by the object.

The result is a nested vector with one element per event name. Each element of this vector is itself a vector of 2-element character vectors. For each event, the first item describes the help message or description (if any) registered for the event and the data type of its result. Each of the remaining elements contains a parameter name and its corresponding data type.

EXAMPLE

Note that if the event does not produce a result, the data type of the result is reported as 'VT_VOID'.

5.37 GetFocus Method 511

Applies To

Description

This method is used to obtain the name of the object that currently has the input focus.

The GetFocus method is niladic.

The result is a simple character vector. An empty result indicates that no Dyalog APL GUI object has the input focus.

See also: *GetFocusObj*.

5.38 GetFocusObj Method 509

Applies To

Description

This method is used to obtain a ref to the object that currently has the input focus.

The `GetFocusObj` method is niladic.

The result is a ref. If there is no Dyalog APL GUI object with the input focus, the result is `⍬NULL`.

See also: *GetFocus*.

5.39 GetItemHandle Method 313

Applies To: *TreeView*

Description

This method is used to obtain the window handle of a particular item in a *TreeView* object.

The argument for `GetItemHandle` is a single item as follows:

[1]	Item number	Integer.

Item number is the index of the item concerned.

The result is an integer containing the window handle of the item.

5.40 GetItemPosition Method 323

Applies To: *ListView*

Description

This method is used to obtain the position of a particular item in a *ListView* object.

The argument for `GetItemPosition` is a single item as follows:

[1]	Item number	Integer

Item number is the index of the item concerned.

The result is a 2-element vector containing the position of the item.

5.41 GetItemState Method 306

Applies To: *ListView*, *TreeView*

Description

This method is used to obtain the status of a particular item in a *ListView* or *TreeView* object.

The argument for `GetItemState` is a single item as follows:

[1]	Item number	Integer

Item number is the index of the item concerned. Be aware that this is index origin dependent.

The result indicates the state of the item as the sum of one or more of the following codes:

-1	Error (most likely the Item number is invalid)
1	Item has the focus
2	Item is selected
8	Item is highlighted for dropping
16	Item is displayed in bold text
32	Item is expanded
64	Item is or has been expanded
4096	Item is checked. See <i>CheckBoxes</i>

5.42 GetMethodInfo Method 552

Applies To: *OCXClass*, *OLEClient*

Description

This method is used to obtain information about a particular method or set of methods supported by a COM object.

For each method supported by a COM object the author will have registered a help message or description of the method (this is in fact optional), the data type of its result (if it has a result), and the name and data type of each of the parameters that must be supplied when you invoke it. The `GetMethodInfo` method returns this information.

The argument to `GetMethodInfo` is a single item as follows:

[1]	Method name(s)	see below

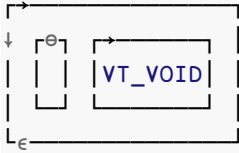
Method name(s) is a simple character vector or a vector of character vectors specifying one or more names of methods supported by the object.

The result is a nested vector with one element per method name. Each element of this vector is itself a vector of 2-element character vectors. For each method, the first item describes the help message or description (if any) registered for the method and the data type of its result. Note that if the event does not produce a result, the data type of the result is reported as 'VT_VOID'. Each of the remaining elements contains a parameter name and its corresponding data type.

EXAMPLE

```
CLNAME←'Microsoft Multimedia Control, Version 6.0'
'MM' WC 'OCXClass' CLNAME
MM.MethodList
AboutBox Refresh OLEDrag

DISPLAY ↑ MM.GetMethodInfo 'AboutBox'
```



5.43 GetMinSize Method 275

Applies To: *Calendar*

Description

This method is used to obtain the minimum size that you must specify for a *Calendar* object for it to display a complete month.

The GetMinSize method is niladic.

The (shy) result of the method is a 2-element numeric vector containing the minimum height and width required for the object to display a complete month.

5.44 GetParentItem Method 312

Applies To: *TreeView*

Description

This method is used to obtain the index of the parent of a particular item in a *TreeView* object.

The argument for GetParentItem is a single item as follows:

[1]	Item number	Integer.

Item number is the index of the item concerned.

The result is an integer containing the index of the parent item.

5.45 GetPropertyInfo Method 550

Applies To: *OCXClass, OLEClient*

Description

This method is used to obtain information about a particular property or set of properties supported by a COM object.

For each property supported by a COM object, the author will have registered the property name, its data type, and an optional help message or description of the property. GetPropertyInfo returns this information.

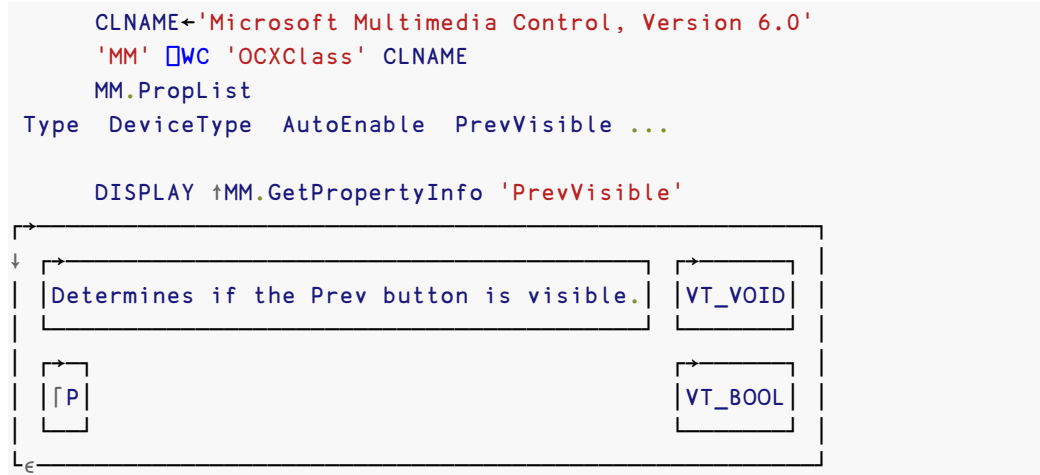
The argument to GetPropertyInfo is a single item as follows:

[1]	Property name(s)	see below

Property name(s) is a simple character vector or a vector of character vectors specifying one or more names of properties supported by the object.

The result is a nested vector with one element per property name. Each element of this vector is itself a 2-element vector of character vectors containing the data type and help message for the corresponding property.

EXAMPLE



If the data type of a property is VT_USERDEFINED, it means that the property may assume one of a set of values defined by a type list. In this case, the name of the type list is returned in place of the string "VT_USERDEFINED". Further information can be obtained using *GetTypeInfo* with this name as a parameter.

5.46 GetServiceState Method 92

Applies To: *Root*

Description

This method is used to obtain the current state of a Dyalog APL service running under Windows. See *Installation/Configuration: Apl Application As A Service*.

The *GetServiceState* method is niladic.

The result of the method is a 7-element numeric vector corresponding to the *SERVICE_STATUS* structure which is described in C++ as follows:

```
typedef struct _SERVICE_STATUS {
    DWORD dwServiceType;
    DWORD dwCurrentState;
    DWORD dwControlsAccepted;
    DWORD dwWin32ExitCode;
    DWORD dwServiceSpecificExitCode;
    DWORD dwCheckPoint;
    DWORD dwWaitHint;
} SERVICE_STATUS, *LPSERVICE_STATUS;
```

For further details, see the on-line documentation for `SERVICE_STATE` and the function `HashDefine` in the sample workspace `aplservice`.

5.47 GetTextSize Method 146

Applies To

Description

The `GetTextSize` method obtains the size of the bounding rectangle of a text item in a given font. The result is given in the co-ordinate system of the object in question. This method is useful for positioning Text objects.

`GetTextSize` duplicates the functionality of the `TextSize` property. It is recommended that you use `GetTextSize` instead of `TextSize` which may be removed in a future release of Dyalog APL.

The argument to `GetTextSize` is a 1 or 2-element array as follows:

[1]	Text item	character array
[2]	Font name	character vector

When you invoke `GetTextSize` you give the text item in whose size you are interested and, optionally, the name of a Font object. The text item may be a simple scalar, a vector or a matrix. If the Font is omitted, the result is given using the current font for the object in question.

EXAMPLES

```
'F' WC 'Form'
F.GetTextSize 'Hello World'
3.385416667 10.7421875

'FNT1' WC 'Font' 'Arial' 72
F.GetTextSize 'Hello World' '#.FNT1'
18.75 65.4296875

F.Coord←'Pixel'
F.FontObj←'FNT1'
F.GetTextSize 'Hello World'
16 77
```

5.48 GetTypeInfo Method 553

Applies To: OCXClass, OLEClient

Description

This method is used to obtain information about a type list supported by a COM object.

The argument to GetTypeInfo is a single item as follows:

[1]	Type List name(s)	see below
[2]	Value	(usually) numeric
[3]	Description	character vector

Type List name(s) is a simple character vector or a vector of character vectors specifying one or more names of type lists supported by the object.

The result is a nested vector with one element per Type List. Each element of this vector is itself a 3-element vector of character vectors made up as follows:

[1]	Name of Constant	character vector
[2]	Value	(usually) numeric
[3]	Description	character vector

5.49 GetVisibleRange Method 262

Applies To: *Calendar*

Description

This method is used to obtain the range of dates that is currently visible in a *Calendar* object.

The GetVisibleRange method is niladic.

The result is a 2-element integer vector containing the first and last dates currently displayed by the object, reported as *IDNs*.

5.50 GetZoomLevel Method 838

Applies To: *HTMLRenderer*

Description

This method is used to retrieve the current CEF ZoomLevel of the HTMLRenderer.

See *SetZoomLevel*.

5.51 GreetBitmap Method 138

Applies To: *Root*

Description

This method is used to display or remove a bitmap, typically during initialisation of a Dyalog runtime application.

The argument to GreetBitmap is $\emptyset$ or a 2 element vector as follows:

[1]	Display	0 = off, 1 = on.
[2]	Bitmap file name	Character vector.

If the argument is $\emptyset$, the bitmap is removed.

The image can also be displayed initially by setting parameter: **greet_bitmap** on the command line. For example:


```
c:\myapp\dyalogrt greet_bitmap=mylogo myws
```

The image is displayed until either an untrapped error occurs, causing the interpreter to (attempt to) display the session window, or the GreetBitmap method is called.

5.52 IDNToDate Method 263

Applies To: *Calendar, DateTimePicker, Root*

Description

This method is used to convert a date from an *IDN* into $\square$ TS format (year, month, day). The corresponding day of the week is also obtained.

The argument to IDNToDate is a single item as follows:

[1]	IDN	Integer

The result is a 4-element integer vector containing the year, month, day, and weekday corresponding to the IDN that was specified.

The value of the 4th element, weekday, is an integer in the range 0-6 that specifies on which day of the week the specified date falls (0=Monday).

EXAMPLE

```
F.C.IDNToDate 36048
1998 9 11 4
```

5.53 ListTypeLibs Method 520

Applies To: *Root*

Description

The ListTypeLibs method reports the names and CLSIDs of all the loaded Type Libraries.

The ListTypeLibs method is niladic.

The result is a nested vector with one element per loaded Type Library.

Each element is a vector of 2-element character vectors. The first is the name of the Type Library; the second is its class identifier or CLSID.

EXAMPLE

```
'EX'WC'OLEClient' 'Excel.Application'
ρListTypeLibs
3
  ↑>ListTypeLibs
Microsoft Excel 9.0 Object Library
{00020813-0000-0000-C000-000000000046}

  ↑>"ListTypeLibs
Microsoft Excel 9.0 Object Library
Microsoft Visual Basic for Applications Extensibility 5.3
Microsoft Office 9.0 Object Library
```

5.54 LockColumns Method 227

Applies To: *Grid*

Description

This method is used to lock one or more columns of a *Grid* object. However, LockColumns is not supported in combination with hierarchical column titles as specified by the *ColTitleDepth* property.

The argument to LockColumns is a 1 or 2-element vector as follows.

[1]	Column(s)	integer scalar, vector or matrix
[2]	Lock flag	0 or 1

Column(s) may be a scalar or a vector specifying the column or columns to be locked or unlocked. Alternatively, it may be a matrix whose first row specifies the columns to be locked and whose second row specifies *where* they are to be locked.

If the *Lock flag* is 1, the corresponding columns are locked. This is the default and may be omitted. If the *Lock flag* is 0, the corresponding columns are unlocked

EXAMPLES

```

F.G.LockColumns 3      ALock 3rd column
F.G.LockColumns 3 0    AUnlock 3rd column
F.G.LockColumns (4 5)  ALock 4th & 5th cols
F.G.LockColumns (2 1p8 4) ALock 8 at 4
F.G.LockColumns 3      ALock 3rd column
F.G.LockColumns 3 0    AUnlock 3rd column
F.G.LockColumns (4 5)  ALock 4th & 5th cols
F.G.LockColumns (2 1p8 4) ALock 8 at 4

```

The result is an integer matrix containing the indices of all locked columns and the positions at which they are currently locked.

The expression:

```
F.G.LockColumns <Θ
```

may therefore be used to obtain the indices of the locked columns, and:

```
F.G.LockColumns(F.G.LockColumns <Θ) 0
```

unlocks all currently locked columns.

Locks are additive. If column 4 is locked, locking column 5 results in both columns 4 and 5 being locked.

A locked column remains fixed in position and does not scroll sideways. The user may enter and edit cells in a locked column in the normal way, but the behaviour of the various cell movement keys (Tab, left and right cursor, and so forth) differs when a locked column is encountered. As a general rule, if a keystroke attempts to move the cursor into a locked column from an adjacent column, and the adjacent column has been scrolled, it is unscrolled and the cursor remains in the (new) column adjacent to the fixed column. If not, the cursor moves into the locked column.

When you lock a column, the position you specify for it to be locked at is a position *in the data* and not the physical position of the column as displayed in the *Grid*. The physical column in the *Grid* depends upon the value of the *Index* property at the time it was locked.

If *c* is the value specified for where a given column is to be locked, the value of the physical column *P* at which it will be displayed in the Grid named *GRID* is:

```
P←C-(2>GRID □WG 'Index')-□IO
```

Furthermore, the position of a locked column given by the result of the LockColumns method changes (with the *Index* property) as the *Grid* is scrolled.

5.55 LockRows Method 226

Applies To: *Grid*

Description

This method is used to lock one or more Rows of a *Grid*. However, LockRows is not supported in combination with hierarchical row titles as specified by the *RowTitleDepth* property.

The argument to LockRows is a 1 or 2-element vector as follows.

[1]	Row(s)	integer scalar, vector or matrix
[2]	Lock flag	0 or 1

Row(s) may be a scalar or a vector specifying the row or rows to be locked or unlocked. Alternatively, it may be a matrix whose first row specifies the data rows to be locked and whose second row specifies *where* in the Grid they are to be locked.

If the *Lock flag* is 1, the corresponding rows are locked. This is the default and may be omitted. If the *Lock flag* is 0, the corresponding rows are unlocked.

EXAMPLES

```
F.G.LockRows 3           aLock 3rd row
F.G.LockRows 3 0         aUnlock 3rd row
F.G.LockRows (4 5)       aLock 4th and 5th rows
F.G.LockRows (2 1p8 4)   aLock row 8 at 4
F.G.LockRows 3           aLock 3rd row
F.G.LockRows 3 0         aUnlock 3rd row
F.G.LockRows (4 5)       aLock 4th and 5th rows
F.G.LockRows (2 1p8 4)   aLock row 8 at 4
```

The result is an integer matrix containing the indices of all locked rows and the positions at which they are currently locked.

The expression:

```
F.G.LockRows <=0
```

may therefore be used to obtain the indices of the locked rows, and

```
F.G.LockRows(F.G.LockRows <0> 0
```

unlocks all currently locked rows.

Locks are additive. If row 4 is locked, locking row 5 results in both rows 4 and 5 being locked.

A locked row remains fixed in position and does not scroll vertically. The user may enter and edit cells in a locked row in the normal way, but the behaviour of the various cell movement keys (Tab, up and down cursor, and so forth) differs when a locked row is encountered. As a general rule, if a keystroke attempts to move the cursor into a locked row from an adjacent row, and the adjacent row has been scrolled, it is unscrolled and the cursor remains in the (new) row adjacent to the fixed row. If not, the cursor moves into the locked row.

When you lock a row, the position you specify for it to be locked at is a position *in the data* and not the physical position of the column as displayed in the Grid. The physical column in the Grid depends upon the value of the Index property at the time it was locked.

If R is the value specified for where a given row is to be locked, the value of the physical row P at which it will be displayed in the Grid named GRID is given by the expression:

```
P←R-(→GRID ⚡WG 'Index')-⚡IO
```

Furthermore, the position of a locked row given by the result of the LockRows method changes (with the Index property) as the Grid is scrolled.

5.56 MDIArrange Method 112

Applies To: *MDIClient*

Description

This method causes the *MDIClient* object to organise the icons associated with any minimised child *Forms* into regimented rows and columns. To permit the user to carry out this action, it is recommended that a suitable callback function or expression is attached to a *MenuItem* or *Button*. The callback function or expression should then call *MDIArrange*.

The *MDIArrange* method is niladic.

5.57 MDICascade Method 110

Applies To: *MDIClient*

Description

This method causes the *MDIClient* object to organise its child *Forms* in an overlapping fashion. To permit the user to carry out this action, it is recommended that a suitable callback function or expression is attached to a *MenuItem* or *Button*. The callback function or expression should then call MDICascade.

The MDICascade method is niladic.

5.58 MDITile Method 111

Applies To: *MDIClient*

Description

This method causes the *MDIClient* object to organise its child *Forms* as a row or column. To permit the user to carry out this action, it is recommended that a suitable callback function or expression is attached to a *MenuItem* or *Button*. The callback function or expression should then call the MDITile method.

Note that because there are restrictions concerning the minimum height and width of a window, MS-Windows does not necessarily respond as requested. If the *MDIClient* is itself of insufficient size, or if it contains a large number of child *Forms*, Windows may choose to tile the *Forms* in a row when a column was specified or vice versa. It may also choose to ignore the event entirely.

The argument to MDITile is 0, or a single item as follows:

[1]	Tile Mode	0 (vertical) 1 (horizontal)

If the argument is 0, the *Tile Mode* defaults to 0.

5.59 MakeGIF Method 261

Applies To: *Bitmap*

Description

This method is used to generate a GIF representation of a picture from a Bitmap object suitable for display by a Web browser.

The MakeGIF method is niladic.

The result is an integer vector containing the encoded GIF image.

EXAMPLE

```
ρGIF←BM.MakeGIF
19620
```

5.60 MakePNG Method 260

Applies To: *Bitmap*

Description

This method is used to generate a PNG (Portable Network Graphics) representation of a picture from a Bitmap object suitable for display by a Web browser.

The MakePNG method is niladic.

The result is an integer vector containing the encoded PNG image.

Example:

```
ρPNG←BM.MakePNG
4930
```

5.61 NameFromHandle Method 136

Applies To: *Root*

Description

This method is used to obtain the name of a particular object from the value of its Handle property.

The argument to NameFromHandle is a single item as follows:

[1]	Handle	The value of the Handle property from an existing object.

The result is a character vector containing the name of the object.

5.62 NewPage Method 102

Applies To: *Printer*

Description

This method causes a *Printer* to start a new page

The NewPage method is niladic.

If you attach a callback function to this event and have it return a value of 0, the page throw will not occur.

5.63 OLEAddEventSink Method 540

Applies To: *OCXClass, OLEClient*

Description

This method connects a named event sink to a COM object and adds the events defined by that event sink to the *EventList* property of the associated namespace.

The argument to OLEAddEventSink is a single item as follows:

[1]	Event sink name	character vector

The result is a number that represents the handle of the event sink. This may be subsequently required.

5.64 OLEDeleteEventSink Method 541

Applies To: *OCXClass, OLEClient*

Description

This method disconnects a named event sink from a COM object and removes the events defined by that event sink from the *EventList* property of the associated namespace.

This method may be used to remove an event sink that was established automatically when the OLE object was created.

The argument to `OLEDeleteEventSink` is a single item as follows:

[1]	Event sink name	character vector

5.65 `OLEListEventSinks` Method 542

Applies To: *OCXClass*, *OLEClient*

Description

This method returns the names of *event sinks* that are currently connected to a COM object.

The list contains the names of all the event sinks that were connected automatically when the object was created, together with any that you have added subsequently using *OLEAddEventSink*.

The `OLEListEventSinks` method is niladic.

The result is a vector of character vectors containing the names of the event sinks connected to the object.

5.66 `OLEQueryInterface` Method 543

Applies To: *ActiveXContainer*, *OLEClient*

Description

This method is used to obtain the methods and properties associated with a particular *interface* that is provided by a COM object. An interface is simply a pointer to a table of methods (not properties) that are exported by an object.

Note that methods and properties exported using the standard `IDispatch` interface are established automatically when the object is created. `OLEQueryInterface` is required only to support alternative or additional interfaces that the object may implement.

The argument to `OLEQueryInterface` is a single item as follows:

[1]	Interface name	character vector

The result is a namespace.

It is normal, although not strictly required, that the new namespace be a child of the one for which the method is run.

Note that if the object does not support a type library, the new namespace will be empty and you will have to establish functions corresponding to the methods exported by the interface using *SetMethodInfo*.

5.67 OLERegister Method 530

Applies To: *OLEServer*

Description

This method is used to register an *OLEServer* object and may be used to install Dyalog OLE Servers as part of a run-time installation.

If the argument to the OLERegister method is a simple character vector, this is treated as text to be inserted into the command line argument for the interpreter. All other arguments are silently ignored.

Note

It is currently necessary to run the APL from which this method is called with Administrator privileges.

5.68 OLEUnregister Method 531

Applies To: *OLEServer*

Description

This method is used to unregister an *OLEServer* object that has previously been saved by Dyalog APL.

The OLEUnregister method is niladic.

This method removes all traces of the object from the Windows registry and erases its Type Library file.

Note that the name of the object removed from the registry is the name of the *OLEServer* object prefixed by the string "dyalog."

5.69 Print Method 100

Applies To: *Printer*

Description

This method causes any spooled output to be printed.

The Print method is niladic.

If you attach a callback function to this event and have it return a value of 0, the printout will not be spooled.

5.70 PrintToPDF Method 845

Applies To: *HTMLRenderer*

Description

This method writes the content displayed in an *HTMLRenderer* object to a specified file in Portable Document Format (pdf).

The argument to PrintToPDF is a simple character scalar or vector containing a file name. Note that the method does not add any extension to the file name that is supplied.

The method returns a Boolean result which indicated whether or not the operation succeeded. If the file name contains a directory path, the path must already exist. The user must have permission to write the file.

5.71 ProgressStep Method 250

Applies To: *ProgressBar*

Description

This method is used to increment the thumb in a *ProgressBar* object.

The *ProgressStep* method is niladic.

The *ProgressStep* method causes the *ProgressBar* to attempt to increment its thumb by the value of its *Step* property, taking into account the settings of its *Limits* and *Wrap* properties.

If the values of the *Thumb*, *Step* and *Limits* properties are THUMB, STEP and LIMITS respectively, the new value of Thumb (and the corresponding position of the highlighted bar) is:

if Wrap is 0:

```
LIMITS[2][THUMB+STEP
```

if Wrap is 1:

```
LIMITS[1]+(1+LIMITS[2]-LIMITS[1])|THUMB+STEP-LIMITS[1]
```

5.72 RTFPrint Method 461

Applies To: *RichEdit*

Description

This method is used to print the contents (*RTFText*) of a *RichEdit* object.

The argument to *RTFPrint* is $\emptyset$, or a 1 to 4-element array as follows:

[1]	Printer name	Optional - character vector (see below)
[2]	Print range	Optional - (see below)
[3]	Number of copies	Optional - Integer.
[4]	Collate	Optional - 0 or 1

Printer name may be the name of an existing Printer object, or the (Windows) name of an installed printer. If you use the latter, the document will be spooled immediately. An empty vector implies the default printer.

Print range may be a simple character vector containing 'All', 'Pages', or 'Selection'. Alternatively, it may be a 3 or 4-element nested vector containing:

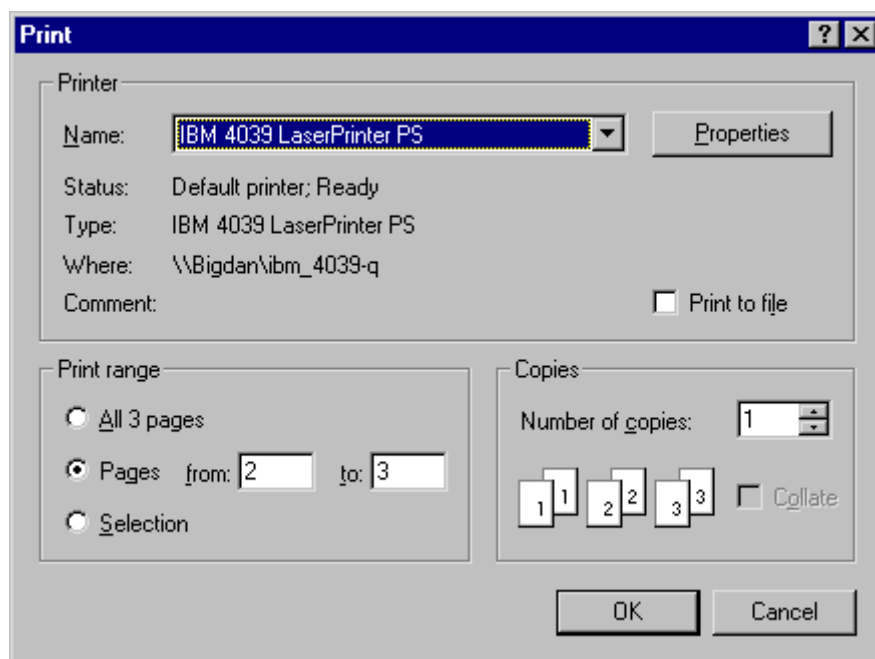
[1]	'All', 'Pages', Or 'Selection'
[2]	Start page (integer)
[3]	End page (integer)
[4]	Maximum pages (ignored)

5.73 RTFPrintSetup Method 460

Applies To: *Printer, RichEdit*

Description

This method is used to display a print set-up dialog box. The dialog box allows the user to select a particular printer, the pages to be printed and other information. The user's choices are returned in the result.



The argument to RTFPrintSetup is θ , or a 1 to 3-element array as follows:

[1]	Print range	Optional - (see below)
[2]	Number of copies	Optional - Integer.
[3]	Collate	Optional - 0 or 1

Print range may be a simple character vector containing 'All', 'Pages', or 'Selection'.

Alternatively, it may be a 3 or 4-element nested vector containing:

[1]	'All', 'Pages', or 'Selection'
[2]	Start page (integer)
[3]	End page (integer)
[4]	Maximum pages

Maximum pages (4th element of *Print range*) may be an integer number, or the name of a reference object. The latter allows the system to calculate the total number of pages required. If the object to which the RTFPrintSetup event is sent is a *RichEdit*, this is the name of a printer object. If the object to which the RTFPrintSetup event is sent is a *Printer*, this is the name of a *RichEdit* object. Both are required because the number of pages of a printed document is dependent upon both the content of the document and the characteristics of the device upon which it will be printed.

If the user presses OK, the result is a 4-element vector containing the user's choices as follows:

[1]	Printer name	character vector
[2]	Print range	(see above)
[3]	Number of copies	Integer
[4]	Collate	0 or 1

EXAMPLE

```
F.T.RTFPrintSetup ('ALL' 1 1 'PR')
IBM 4039 LaserPrinter PS    Pages 2 3 3 1 0
```

5.74 RowChange Method 158

Applies To: *Grid*

Description

This method is used to change the data in a row of a *Grid* object.

The argument to `RowChange` is a 2-element array as follows.

[1]	Row number	integer
[2]	Row data	array

Row data must be a scalar or a vector whose length is equal to the number of columns in the Grid. Its elements may be scalar numbers, character vectors or matrices.

5.75 RowSetVisibleDepth Method 173

Applies To: *Grid*

Description

This method is used to set the maximum visible depth of data in rows of a *Grid*.

The argument to `RowSetVisibleDepth` is a numeric scalar as follows

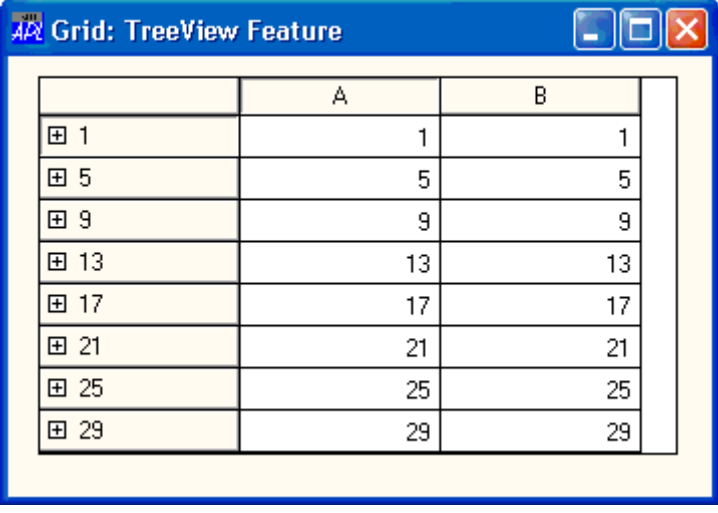
[1]	Depth	integer

All rows in the grid that have a value of *RowTreeDepth* less than or equal to *Depth* are expanded. Rows with a value of *RowTreeDepth* greater than *Depth* are collapsed.

Note: *Expanding* and *Retracting* events are not generated when this method is called.

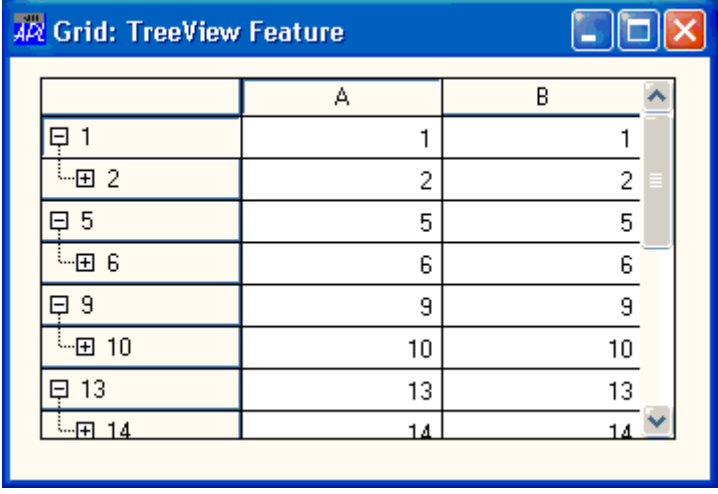
EXAMPLES

```
'F'[]WC'Form' 'Grid: TreeView Feature'
'F.G'[]WC'Grid'(30 2p2/130)
F.G.RowTreeDepth←30p0 1 2 2
```



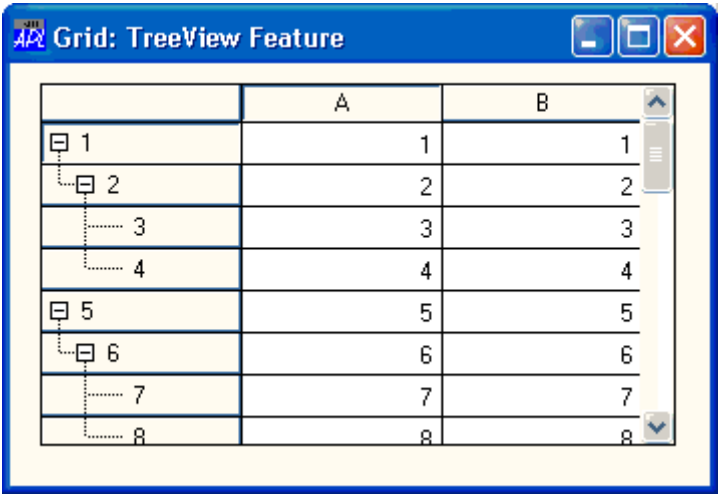
	A	B
1	1	1
5	5	5
9	9	9
13	13	13
17	17	17
21	21	21
25	25	25
29	29	29

F.G.RowSetVisibleDepth 1



	A	B
1	1	1
2	2	2
5	5	5
6	6	6
9	9	9
10	10	10
13	13	13
14	14	14

F.G.RowSetVisibleDepth 99



5.76 SetCellSet Method 171

Applies To: *Grid*

Description

The SetCellSet method sets the value of the *CellSet* property of a *Grid* for a particular cell.

The argument to SetCellSet is a 3-element array as follows:

[1]	Row	integer
[2]	Column	integer
[3]	Value	0 or 1

5.77 SetCellType Method 156

Applies To: *Grid*

Description

This method is used to change the type of a particular cell in a *Grid*.

The argument to SetCellType is a 3-element vector as follows :

[1]	Cell row	integer
[2]	Cell column	integer
[3]	Cell type	integer

5.78 SetEventInfo Method 547

Applies To: *ActiveXControl*, *OLEServer*

Description

This method is used to register an event that may be generated by an *ActiveXControl* or *OLEServer* object.

A host application that wishes to attach a callback function to an event in a Dyalog APL *ActiveXControl* or *OLEServer*, needs to know the name of the event and the number and data types of any parameters that the event may supply. It also needs to know the data type (if any) of the result that the callback function may be expected to pass back to the control.

An *ActiveXControl* or *OLEServer* generates an event in the host application using `⋄ ⌈NQ`. The right argument is a vector whose first 2 elements are character vectors containing the names of the *ActiveXControl* or *OLEServer* and the event respectively. The parameters for the event are passed as additional elements in the argument.

Another way to think about it is that when you generate an event using `⋄ ⌈NQ`, you are effectively calling a function, of your specification, in the host application. To enable the host application to accept the function call, it needs to know the number of parameters that you will supply and their data types.

A further consideration is that if you specify that the data type of a parameter is a *pointer* (for example, `'VT_PTR TO I4'`) this will allow a callback function to modify the parameter in-situ. If so, the result returned by `⋄ ⌈NQ` will be the modified values of any such parameters; this is a similar mechanism to `⌈NA`.

The argument to `SetEventInfo` is a 1, 2, 3 or 4-element array as follows:

[1]	Event name	character vector
[2]	Event info	nested array (see below)
[3]	Help ID	integer
[4]	DISPID	integer. See <i>DISPID (Dispatch ID)</i>

Event info

Event info, specifies an optional help string which describes what the event does, the data type of the result (if any) and the names and data types of its arguments.

If the event is fully described, each element of Event Info is a 2-element vector of character vectors. The first element contains the help string and the *COM data type* of the result that the callback function in the host application is expected to supply. Subsequent elements contain the name and *COM data type* of each of the parameters supplied by the event.

However, both the help string and the names of the parameters are optional and may be omitted. If so, one or more elements of *Event Info* may be a simple character vector.

Help ID

This is an integer value that identifies the help context id for the event within the help file associated with the *HelpFile* property of the *ActiveXControl* object. The value `-1` means that no help is provided. APL stores this information in the registry from where it may be retrieved by the host application.

EXAMPLE

The example *Dual* *ActiveXControl*, that is fully described elsewhere, generates a *ChangeValue1* event. This event occurs when the user moves the thumb in a *TrackBar* that is internal to an instance of the *ActiveXControl*.

The external *ChangeValue1* event is fired by an internal APL callback function (called *ChangeValue*) that is attached to *ThumbDrag* and *Scroll* events on the *TrackBar* object. The internal callback function is :

```

[0]  ChangeValue MSG
[1]  A Callback for ThumbDrag and Scroll
[2]  Value1←4 □NQ'' 'ChangeValue1'(>1↑MSG)
[3]  CalcValue2
[4]  'V1'□WS'Text' (▯Value1)
[5]  'V2'□WS'Text' (▯Value2)

```

Note that `ChangeValue[2]` generates the external `ChangeValue1` event by invoking `4 □NQ`, passing it the new value provided by the `TrackBar`. However, the host application is permitted to modify that value, returning it in the result of `4 □NQ`. This result, rather than the `TrackBar` value itself, is then used to update other (Label) controls in the object.

The following statements were used to declare the *ChangeValue1* event. The event provides a single parameter named *Value1* that may be modified in-situ by a callback function in the host application. The callback is not, otherwise, expected to return a result.

```

INFO←▯'Occurs when the value of the control is changed' 'VT_VOID'
INFO,←▯'Value1' 'VT_PTR TO VT_I4'
F.Dual.SetEventInfo 'ChangeValue1' INFO

```

If the host application was Visual Basic, a suitable callback function might be:

```

Private Sub Dual1_ChangeValue1(Value1 As Long)
Value1=2*(Value1\2)
End Sub

```

This callback function receives the proposed new value of the control as the parameter `Value1`, and modifies it, forcing it to be an even number.

5.79 SetFinishText Method 366

Applies To: *PropertySheet*

Description

The `SetFinishText` method sets the caption of the Finish button in a Wizard-style *PropertySheet*.

The argument to `SetFinishText` is a single item as follows:

[1]	Finish button text	character vector

5.80 SetFnInfo Method 545

Applies To: *ActiveXControl*, *OLEServer*

Description

This method is used to describe an APL function that is to be exported as a method, a Property Get Function, or a Property Put Function of an *ActiveXControl* or *OLEServer* object.

An exported function must be a niladic or monadic defined function (dfns and derived functions are not allowed) and may optionally return a result. Ambivalent functions (functions with optional left argument) are allowed, but will be called monadically by the host application.

COM syntax differs from APL syntax in many ways and the *SetFnInfo* method is required to declare an APL function to COM in terms that COM understands. In particular, although monadic APL functions take just one argument, COM functions may take several parameters, and some may be optional.

A function exported by *SetFnInfo* will be called by a host application with the number of parameters that *SetFnInfo* has described. The argument received when the function is called by a host application, will be a nested vector of this length.

The argument to *SetFnInfo* is a 2, 3, 4, 5 or 6-element array as follows:

[1]	Function name	character vector
[2]	Function info	nested array (see below)
[3]	Help ID	integer
[4]	Function type	integer
[5]	Property name	character vector
[6]	DISPID	integer. See <i>DISPID (Dispatch ID)</i>

Function info

This specifies an optional help string which describes what the function does, the data type of the result (if any) and the names and data types of its arguments.

If the function syntax is fully described, each element of *Function Info* is a 2-element vector of character vectors. The first element contains the help string and the COM

data type of the function's result. Subsequent elements contain the name and COM *data type* of each parameter.

However, both the help string and the names of the parameters are optional and may be omitted. If so, one or more elements of *Function Info* may be a simple character vector.

Consider a very basic function `ADD` in an *ActiveXControl* called `F.dbase`, that is designed to add a record to a personnel database. The database consists only of a list of names, ages and addresses.

Function `ADD` expects to be called with a name (character string), age (number) and address (character string), and returns a result 0 or 1 (Boolean) according to whether the record was successfully added. This function could be declared as follows:

```
HELP←'Adds a new record to the personnel database'
SPEC←c(HELP 'VT_BOOL')      A Result is Boolean
SPEC,←c('Name' 'VT_BSTR')   A 1st param called 'Name' is a
string
SPEC,←c('Age' 'VT_I4')      A 2nd param called 'Age' is an
integer
SPEC,←c('Address' 'VT_BSTR')A 3rd param called 'Address' is a
string

F.dbase.SetFnInfo 'ADD' SPEC
```

Alternatively, but much less helpfully, the function could be declared to take a single unnamed nested argument, leaving it to the host application programmer to guess at its structure :

```
SPEC←c(' ' 'VT_BOOL')      A No help string, result is
Boolean
SPEC,←c(' ' 'VT_ARRAY OF VT_VARIANT') A Param is a nested array
F.dbase.SetFnInfo 'ADD' SPEC
```

Help ID

This is an integer value that identifies the help context id within the help file associated with the `HelpFile` property of the *ActiveXControl* object. The value `-1` means that no help is provided. APL stores this information in the registry from where it may be retrieved by the host application.

Function type

This specifies the type of function being exported. This is an integer with one of the following values:

1	Function is a method
2	Function is a property get function
4	Function is a property put function

In both these last two cases, the name of the property, which is totally independent of the name of the APL function, is given as *Property name*.

If omitted, the function type is *method*.

5.81 SetItemImage Method 315

Applies To: *TreeView*

Description

This method is used to allocate a picture icon to a particular item in a *TreeView* object.

The argument to *SetItemImage* is a 2-element array as follows:

[1]	Item number	Integer.
[2]	Picture index	Integer.

Item number is the index of the item concerned.

Picture index is an index into the array of bitmapped images in the corresponding *ImageList* object which is referenced via the *ImageListObj* property.

5.82 SetItemState Method 307

Applies To: *ListView*, *TreeView*

Description

This method is used to set the status of a particular item in a *ListView* or *TreeView* object.

The argument to `SetItemState` is a 2-element array as follows:

[1]	Item number	Integer. The index of the item concerned.
[2]	Status	Integer

The status of an item is calculated as the sum of one or more of the following state codes:

-1	Error (most likely the Item number is invalid)
1	Item has the focus
2	Item is selected
8	Item is highlighted for dropping
16	Item is displayed in bold text
32	Item is expanded
64	Item is or has been expanded
4096	Item is checked. See <i>CheckBoxes</i>

5.83 SetMethodInfo Method 556

Applies To: *OCXClass*, *OLEClient*

Description

This method is used to redefine the arguments or data types associated with a method that is exported by a COM object. `SetMethodInfo` is used to override the information provided by the object's Type Library.

The argument to `SetMethodInfo` is a 2 or 3-element array as follows:

[1]	Method name	character vector
[2]	Method info	nested vector (see below)
[3]	Method index	integer

If you wish to describe the method completely, the structure of *Method info* should be identical to the structure returned by `GetMethodInfo`, although abbreviated formats are also allowed.

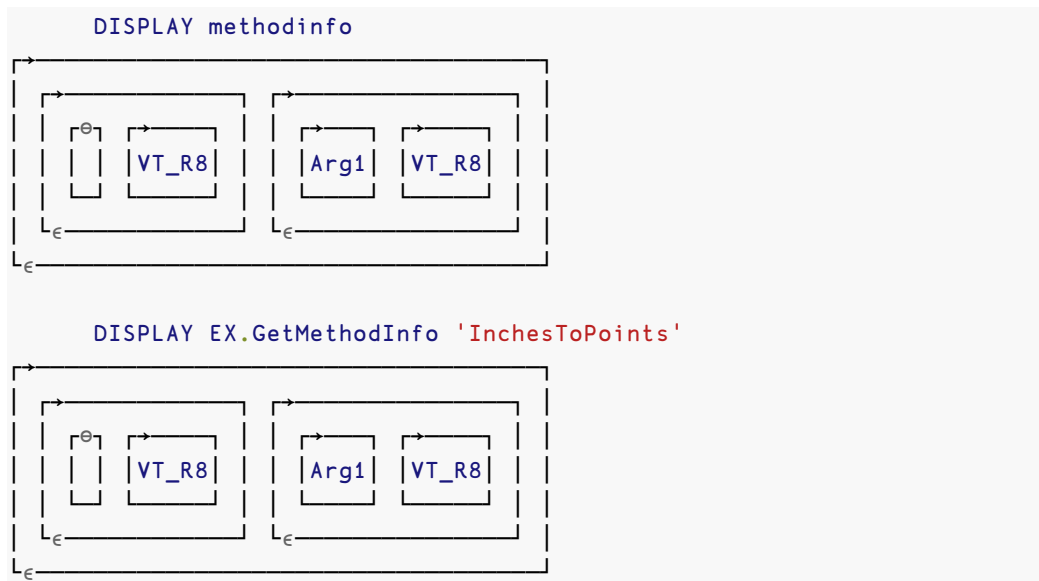
If the object exports the method directly, and not through the standard IDispatch interface, you must also specify *Method Index*, which is the index of the method in the object's virtual table (vtable). This information may be available in printed documentation or in a C-language header file.

For example, the `InchesToPoint` method exported by `Excel.Application` takes a single argument whose name is `Arg1` and whose data type is `VT_R8`. The function returns a result of the same data type. The details provided in the `Excel.Application` Type Library are in fact correct, but if you wanted to redefine them, the following statements could be used to describe the `InchesToPoints` method.

```
'EX' □WC 'OLEClient' 'Excel.Application' ('AutoBrowse' 0)

methodinfo← (' 'VT_R8')('Arg1' 'VT_R8')
EX.SetMethodInfo 'InchesToPoints' methodinfo
```

Note that the structure of variable `methodinfo` is identical to the result of the `GetMethodInfo` method.



Unless you are going to call the method using the names of its arguments, these names are clearly superfluous and may be omitted, for example:

```
methodinfo← 'VT_R8' 'VT_R8'
EX.SetMethodInfo 'InchesToPoints' methodinfo
```

5.84 SetPropertyInfo Method 554

Applies To: *OCXClass*, *OLEClient*

Description

This method is used to redefine a property that is exported by a COM object. SetPropertyInfo is used to override the information provided by the object's Type Library.

The argument to SetPropertyInfo is a 2 or 3-element array as follows:

[1]	Property name	character vector
[2]	Property info	nested vector
[3]	Property function	integer
[4]	DISPID	integer. See <i>DISPID</i> (<i>Dispatch ID</i>)

For example, the Visible property exported by *Excel.Application* has the data type VT_BOOL and may be declared as follows:

```
'EX' [WC 'OLEClient' 'Excel.Application'
EX.SetPropertyInfo 'Visible' 'VT_BOOL'
```

Property function may be required if the property value is retrieved or set via a function. This typically applies if the property takes parameters and will result in the property being fixed as a function rather than as a variable. Such properties may have a PropertyGet function, a PropertyPut function and/or a PropertyPutByReference function. If so, it is necessary to say to which of these three functions the details apply. The value of *Property function* is an integer 2 (PropertyGet), 4 (PropertyPut), or 8 (PropertyPutByReference).

For example, the following statement declares the PropertyGet function for the Item property of the Fields collection of the OLE object DAO.DBEngine. This property takes an index (into the collection) and returns an object.

```
Fields.SetPropertyInfo 'Item' ('VT_DISPATCH' 'VT_I4') 2
```

5.85 SetServiceState Method 93

Applies To: *Root*

Description

This method is used to set the state of a Dyalog APL service running under Windows. See *Installation/Configuration: Apl Application As A Service*.

The argument to the `SetServiceState` method is the desired state of the service. This should be one of the following numeric values:

SERVICE_STOPPED	1
SERVICE_RUNNING	4
SERVICE_PAUSED	7

The state of a Windows service is determined by the user and the Windows Service Control Manager (SCM).

When the SCM requests a change of state, the APL interpreter responds by setting its state to the corresponding *pending* level (SERVICE_STOPPED_PENDING, SERVICE_RUNNING_PENDING or SERVICE_PAUSED_PENDING) and then generates a *ServiceNotification* event.

It is the responsibility of the APL service to process this event, perform the appropriate application tasks, and then respond (to the SCM) by calling the *SetServiceState* method to confirm that the service has reached the desired state.

5.86 SetVarInfo Method 546

Applies To: *ActiveXControl*, *OLEServer*

Description

This method is used to describe an APL variable that is to be exported as a property of an *ActiveXControl* or *OLEServer* object.

The argument to `SetVarInfo` is a 2 or 3-element array as follows:

Variable info is either a simple character vector that specifies the *COM data type* of the variable, or a 2-element vector of character vectors whose first element specifies a help string and whose second element specifies the *COM data type*.

Help ID is an optional integer value that identifies the help context id within the help file associated with the `HelpFile` property of the *ActiveXControl* object. The value `-1`

means that no help is provided. APL stores this information in the registry from where it may be retrieved by the host application.

5.87 SetWizard Event 365

Applies To: *PropertySheet*

Description

If enabled, this event is reported when the user has clicked the Next or Back button in a *PropertySheet* with Style 'Wizard'. This action also generates PageNext (or PageBack) and PageDeactivate and PageActivate events. The SetWizard event is the final event to be reported as a result of this action, and is the only one that is affected by the result of a callback function. The event message reports the active/inactive state of the 3 page changing buttons (Back, Next and Finish) that should result from the action. Note that the Next and Finish buttons occupy the same position and are mutually exclusive.

The event message reported as the result of `WM_SETWIZARD`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'SetWizard' or 365
[3]	Active state Back	0 or 1
[4]	Active state Next	0 or 1
[5]	Active state Finish	0 or 1
[6]	Finish caption	character vector

You may alter the state of the buttons by changing elements [3-5] of the event message and returning it as a result of your callback. You may also set the state of the buttons at any time by calling SetWizard as a method.

When the event is reported by `WM_SETWIZARD`, element 6 is an empty vector. If you modify it and return it in the result of a callback, the caption of the Finish button changes accordingly and the Back and Next buttons disappear. This happens regardless of the states you specified in elements [3-5].

5.88 SetZoomLevel Method

Applies To: *HTMLRenderer*

Description

Sets the CEF ZoomLevel. The default (unzoomed) level is 0. Setting a positive value will increase the zoom, whereas setting a negative will decrease the zoom. The zoom scale is not linear; rather the effective scaling is approximately 1.2*level, so, setting the ZoomLevel to 1 will result in an approximate 20% size increase. ZoomLevel affects all instances of HTMLRenderer windows; it is not possible to have different ZoomLevels for individual windows.

The argument to SetZoomLevel is a single numeric value:

[1]	ZoomLevel	Numeric

EXAMPLES

```
▼ hr Zoom level;lb;in
[1] hr.SetZoomLevel level
[2] level←hr.GetZoomLevel
[3] ((level='-')/level)←'- '
[4] lb←'<label>Zoom Level is </label>'
[5] in←'<input type="number" value="',level,'"></input>'
[6] hr.HTML←lb,in
▼
```

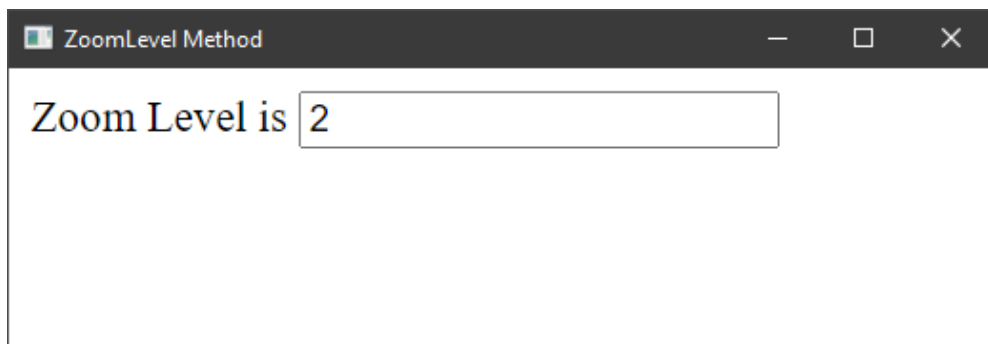
```
hr.WC'HTMLRenderer' ('Caption' 'ZoomLevel Method')
hr Zoom 0
```



```
hr Zoom 1
```



hr Zoom 2



hr Zoom -1



5.89 Setup Method 101

Applies To: *Printer*

Description

This method causes the system to display a standard *Printer Setup* dialog box and thereby allows the user to alter the printer settings. This is a "modal" dialog box that must be closed before the APL application can continue.

The Setup method is niladic.

If you attach a callback function to this event and have it return a value of 0, the dialog box will not appear.

5.90 ShowBalloonTip Method 860

Applies To: *SysTrayItem*

Description

The ShowBalloonTip method displays a BalloonTip in a *SysTrayItem* object.

The argument to ShowBalloonTip is a 1, 2, 3 or 4-element array as follows:

[1]	Title	character vector
[2]	Text	character vector or matrix
[3]	Icon	Integer scalar, a character vector or a ref
[4]	Flags	Integer

The *Title* parameter is the text to be displayed in the BalloonTip title (maximum length 64).

The *Text* parameter is the text (maximum length 256) to be displayed in the BalloonTip. If omitted or empty, the BalloonTip is not displayed.

If the *Icon* parameter is an integer, it means:

0	No icon
1	Information icon
2	Warning icon
3	Error icon

Other values represent the name or a ref to an *Icon* object. If the *Icon* parameter is omitted, no icon is displayed in the BalloonTip.

If the *Icon* parameter specifies a *large* icon object (32 x 32 bits) the *Flags* parameter must be 32. Otherwise this parameter is not used.

5.91 ShowDevTools Method 849

Applies To: *HTMLRenderer*

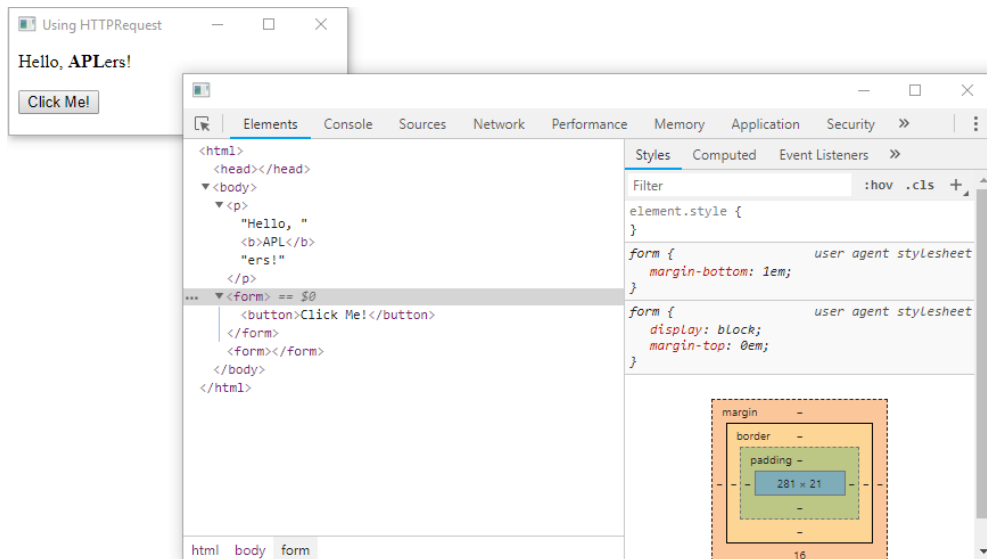
Description

This method displays or hides the CEF development tools window that displays information about the content being displayed by the *HTMLRenderer*.

It takes a Boolean argument 0(hide) or 1(display).

Using the example illustrated in the [HttpRequest](#) topic ...

```
hr.ShowDevTools 1
```



5.92 ShowHelp Method 580

Applies To: *OCXClass*, *OLEClient*

Description

This method is used to display the Windows help file for a COM object or the help topic associated with one of its properties, events or methods.

The argument to ShowHelp is $\emptyset$, or a single item as follows :

[1]	Topic	character vector.

Topic specifies the name of a property, event or method.

In the case of an OLE Control, the object name may be the name of the *OCXClass* or an instance of the *OCXClass*.

5.93 ShowItem Method 316

Applies To: *TreeView*

Description

This method is used to display a particular item in a *TreeView* object.

The argument to ShowItem is a single item as follows:

[1]	Item number	Integer.

Item number specifies the index of the item concerned.

In order to display the requested item, the parent item (if any) will be opened and the object will be scrolled if necessary.

5.94 ShowProperties Method 560

Applies To: *OCXClass*

Description

This method is used to display the PropertySheet for *an instance of* an OLE Control. The user may then modify some or all of the properties of the control by changing values in the property sheet. This facility is intended to be used in the context of a GUI design tool but may also be useful in certain end-user applications.

The ShowProperties method is niladic.

5.95 TCPGetHostID Method 376

Applies To: *Root*, *TCPSocket*

Description

This method is used to obtain the IP Address of your PC.

The TCPGetHostID method is niladic.

The (shy) result is a character string containing your IP address. If you have more than one, it will return the first.

For example:

```
TCPGetHostID
193.32.236.43
```

5.96 TCPSend Method 375

Applies To: *TCPSocket*

Description

This method is used to send data to a remote process connected to a *TCPSocket* object

The argument to TCPSend is a 1 or 3-element array as follows:

[1]	Data	the data to be sent
[2]	IP address	character vector
[3]	Port number	integer

If Style is 'Char', the data to be sent must be a character vector. If Style is 'Raw', the data to be sent must be an integer vector whose elements are in the range -128 to 255. If Style is 'APL', any array may be transmitted.

The optional *IP address* and *Port number* parameters specify the intended recipient of the message and apply only if the *SocketType* is 'UDP', in which case they are mandatory. If the *SocketType* is 'Stream', these parameters will be ignored and should be omitted.

5.97 TCPSendPicture Method 380

Applies To: *TCPSocket*

Description

This method is used to transmit a picture represented by a *Bitmap* object to a TCP/IP socket. The picture may be transmitted in GIF or in PNG format.

The argument to TCPSendPicture is a 1 or 2-element array as follows:

[1]	Bitmap name	character vector
[2]	Picture format	character vector, 'GIF' or 'PNG'

If *Picture format* is omitted, the default is GIF format.

Note that the *Style* of the *TCPSocket* object must be set to 'Raw' before you execute the TCPSendPicture method.

The (shy) result of the method is an integer that reports the number of bytes that were transmitted.

EXAMPLE

```
S1.TCPSendPicture 'BM' 'PNG'  
4930
```

Note

Although PNG is recognised as the latest graphics standard for displaying pictures, not all Web browsers support it.

See also: *MakeGIF*, *MakePNG*

5.98 Undo Method 170

Applies To: *Grid*

Description

This method is used to undo the previous change in a *Grid* object.

The *Grid* object maintains a buffer of the most recent 8 changes made by the user since the *Values* property was last set by *Dyalog APL Language: Wc* or *Dyalog APL Language: Ws*.

Your application can restore these changes one by one by calling the Undo method on the *Grid*. The Undo method restores the most recent change made by the user and removes that change from the undo stack.

It is therefore not possible to "undo an undo".

The argument to Undo is $\emptyset$, or a single item as follows :

[1]	Number of changes	integer

If called with an argument of $\emptyset$, the default value for the *Number of changes* is 1. This restores the most recent change.

5.99 Wait Method 147

Applies To

Description

The Wait method is the same as executing `⌈DQ` on the object.

The Wait method is niladic.

```
'F'⌈WC'Form'
...
Z←F.Wait
```

5.100 WebSocketClose Event 843

Applies To: *HTMLRenderer*

Description

This event is triggered when the *HTMLRenderer* client closes the WebSocket. It is for notification only.

The event message reported as the result of `onclose`, or supplied as the right argument to your callback function, is a 5-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'WebSocketClose' or 843
[3]	ID	Character vector containing the ID of the WebSocket
[4]	Status Code	Integer. 1000 indicates normal closure. See https://www.iana.org/assignments/websocket/websocket.xml#close-code-number .
[5]	Reason	Character vector, maximum length 123 bytes.

When called as a method, the result is 0.

EXAMPLE

```
hr.WebSocketClose '223d0f781e95113'
```

5.101 WebSocketSend Method 847

Applies To: *HTMLRenderer*

Description

This method is used to send data to a WebSocket. The argument to `WebSocketSend` is a 2, 3 or 4-element vector as follows:

[1]	ID	Character vector containing the ID of the WebSocket
[2]	Data	Character or integer vector. Integers must be in the range -128 to 255.
[3]	FIN	Boolean. 1 indicates that this is the last chunk; 0 that there is more to come. This is not currently supported by the CEF and should be set to 1.
[4]	OpCode	Integer. 1 means UTF-8 data, 2 means binary data. Should be 0 if FIN is 0.

The result is 0.

EXAMPLE

```
0 hr.WebSocketSend '5d61d8330065608' 'Hello World'
```

6 (A-Z)

6.1 Abort Method 103

Applies To: [Section 6.455](#)

Description

This method causes the print job to be aborted and all pending output to be discarded.

The Abort method is niladic.

If you attach a callback function to this method and have it return a value of 0, the print job will continue.

6.2 Accelerator Property

Applies To

Description

This property specifies a keystroke that, when pressed by the user, will generate a [Section 6.513](#) event on an object. It applies to all objects whether or not they possess a "natural" [Section 6.513](#) event. You can therefore associate a keystroke with an arbitrary action on any object you desire.

The Accelerator property is a 2-element integer vector. The first element is a key number which is the number by which Windows knows the key. The second element is the shift state which is the sum of 1 (Shift key), 2 (Control key) and 4 (Alt key).

For example, to attach the keystroke Ctrl+A to an object, you would set its Accelerator to (65 2). To attach the keystroke Shift+Ctrl+F1 (key number 112), you would set its Accelerator to (112 3). Key numbers may be obtained by displaying the messages generated by the [Section 6.340](#) event.

Note that a keystroke used as an Accelerator will **not** generate a [Section 6.340](#) event.

6.3 AcceptFiles Property

Applies To

Description

The AcceptFiles property is Boolean and specifies whether or not an object will accept a file drag/drop operation. Its default value is 0. If set to 1, the object will report a [Section 6.178](#) event when file icons are dropped on it.

6.4 ActivateApp Event 139

Applies To: [Section 6.489](#)

Description

If enabled, this event is reported when the user switches to or from a Dyalog APL/W application.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ActivateApp' or 139
[3]	Activation flag	0 or 1

The Activation flag is 0 when the user switches *from* Dyalog APL to another application.

The Activation flag is 1 when the user switches to Dyalog APL from another application.

6.5 Active Property

Applies To

Description

This property specifies whether or not an object is currently responsive to user actions. It is a single number with the value 0 (object is inactive and does not generate events) or 1 (object is active and capable of generating events). The default is 1.

Setting Active to 0 disables the object (and all its children), even though the object may be referenced in the argument to *Dyalog APL Language: Dq*. It is therefore possible to deactivate an object from a callback function.

In general, the text associated with an object whose Active property is 0 is displayed in grey.

6.6 ActiveXContainer Object

Purpose: The ActiveXContainer object represents the application that is currently hosting an instance of an ActiveXControl object.

Parents ActiveXControl

Properties Type Event FontObj FCol BCol Data KeepOnClose MethodList ChildList EventList PropList

Methods Detach OLEQueryInterface

Events Close Create AmbientChanged

Description

An ActiveXContainer is used to represent the host application that is hosting an [Section 6.7](#) object, and provides access to its ambient properties such as font, and colour.

An ActiveXContainer object is created using the [Section 6.124](#) property of the [Section 6.7](#) object.

For example, the following expression, executed within an [Section 6.7](#) instance creates an ActiveXContainer named 'CONT'

```
'CONT' ⍵NS ⍵WG'Container'
```

The ambient properties of the host application are reported by the [Section 6.225](#), [Section 6.205](#) and [Section 6.45](#) properties which are all read-only.

The ActiveXContainer object supports the [Section 6.19](#) event which is reported when any of the ambient properties change. This event allows the ActiveXContainer to react to such changes.

6.7 ActiveXControl Object

Purpose: The ActiveXControl object represents a Dyalog APL namespace as an ActiveX control.

Parents Form

Children ActiveXContainer Animation Bitmap BrowseBox Button ButtonEdit Calendar Circle Clipboard ColorButton Combo ComboEx CoolBar Cursor DateTimePicker Edit Ellipse FileBox Font Form Grid Group HTMLRenderer Icon Image ImageList Label List ListView Locator Marker MDIClient Menu MenuBar Metafile MsgBox OCXClass OLEClient OLEServer Poly Printer ProgressBar PropertySheet Rect RichEdit Scroll Spinner Splitter Static StatusBar SubForm TabBar TabControl TCPSocket Text Timer TipField ToolBar ToolControl TrackBar TreeView UpDown

Properties Type ClassName Posn Size Coord Border Active Visible Event Draggable FontObj FCol BCol Picture CursorObj AutoConf YRange XRange Data TextSize EdgeStyle Handle Translate Accelerator AcceptFiles ClassID Container KeepOnClose HelpFile ToolboxBitmap TypeLibID TypeLibFile LastError Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj SetFnInfo SetVarInfo SetEventInfo

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap AmbientChanged Select PreCreate

Description

The ActiveXControl object represents a Dyalog APL namespace as an ActiveX control.

During development, an ActiveXControl is a container object that is the child of a Form and acts as a wrapper for one or more other GUI objects.

To make an ActiveXControl available to another application, you must select *Make OCX* from the *Session File* menu. This creates an .OCX file that contains your entire workspace and all of the ActiveXControls within it.

Once an ActiveXControl has been saved in an .OCX file, any application that supports ActiveX may create and use instances of it.

When an ActiveX control is loaded by a host application, it and any code that it requires, is loaded into the host application's address space; it does not run in a separate address space.

During development, an ActiveXControl is powered by the development version of Dyalog APL. However, an ActiveXControl object that is loaded by a host application, is powered by a DLL version of Dyalog APL. This automatically gets loaded when a host application creates the first instance of any Dyalog APL ActiveX control. However, within a single host application, other instances of the same or other Dyalog APL ActiveX controls share the same copy of DYALOG.DLL.

Like the development and run-time versions of Dyalog APL, DYALOG.DLL has an active workspace. When an application loads an ActiveXControl, DYALOG.DLL *copies* the *top-level namespace* that owns the ActiveXControl, together with everything it contains, into the active workspace. For example, if the ActiveXControl is named `Controls.Form1.Ctrl1`, the act of creating the first instance of `Ctrl1` will cause the entire contents of the `Controls` namespace to be copied, from the corresponding .OCX file, into the active workspace. This affords the potential for controls from different OCX files to clash, but the name clash conflict is restricted to just one name.

Each instance of an ActiveXControl, is represented by a separate namespace which is automatically *cloned* from the original ActiveXControl namespace. Each instance namespace is entirely separate from any other instance namespace and there is no way for one instance to reference or see any other instance; nor can it reference the original class namespace from which it was cloned. In fact, each instance appears to itself to be the one and only original class namespace. Using the previous example, each instance of `Ctrl1` believes that its full pathname is `#.Controls.Form1.Ctrl1`, although each instance is in fact a separate clone of that namespace.

When an application creates an instance of an ActiveXControl, it does so as the child of some object within its own GUI hierarchy. From the instance's viewpoint, its parent Form is replaced by a different GUI object that imposes position, size, font, background colour, and other ambient properties.

The external name of an ActiveXControl is made up of the character vector defined by the [Section 6.97](#) property, prefixed by the string "Dyalog ", and followed by the string "Control". If [Section 6.97](#) is empty (which is the default), the name of the ActiveXControl namespace is inserted instead. Note that the name should not include APL symbols such as `Δ`. [Section 6.97](#) may only be specified when you create the ActiveXControl with `⌈WC` and may not be changed using `⌈WS`.

The [Section 6.128](#) property is read-only and its value is always 'Pixel'. If you wish to use a different co-ordinate system for the children of an ActiveXControl object, it is necessary to set [Section 6.128](#) separately on each one of them.

[Section 6.452](#) and [Section 6.556](#) are negotiable properties. When an instance of the ActiveXControl is created, the values of [Section 6.452](#) and [Section 6.556](#) will be assigned by the host application. You may change these values using `OWS`, but the host application has the right to refuse them and there is no guarantee that you will get what you set.

The [Section 6.49](#) and [Section 6.184](#) properties may be used to control the outline appearance of the ActiveXControl object.

The [Section 6.173](#) and [Section 6.338](#) properties apply only during development and are otherwise ignored.

The [Section 6.622](#) property specifies the name of a [Section 6.47](#) object that may be used by a host application to represent the ActiveXControl when its complete visual appearance is not required.. For example, if you add an ActiveX control to the Microsoft Visual Basic development environment, its bitmap is added to the toolbox. The Bitmap should therefore be of an appropriate size, usually 24 x 24 pixels.

The [Section 6.124](#) property provides access to an [Section 6.6](#) object that represents the host application itself. This may be used to obtain the values of ambient properties, or to access methods exposed by the host application via OLE interfaces.

When an instance of an ActiveXControl is created, it generates first a [Section 6.453](#) event, and then a Create event. The [Section 6.453](#) event is generated at the point the *instance* is made.

The [Section 6.131](#) event is generated at the point when the host application requires the instance to appear visually. If, as is recommended, you create child controls of the instance when it is created, you must respond to the [Section 6.131](#) event, because at the time that [Section 6.453](#) is generated, the object does not have a window.

Host applications which support two different modes of operation, namely design mode and run mode, differ in the way that they create instances of ActiveX controls. Microsoft Access does not require an ActiveX control to appear properly in design mode. Instead, it draws a simple box containing just the name of the object. If your ActiveXControl is hosted by Microsoft Access, it will get a [Section 6.453](#) Event when an instance is created in design mode, and a [Section 6.131](#) event only when it enters run mode. Microsoft Visual Basic, however, requires the object to draw itself immediately, even in design mode, and so a [Section 6.131](#) event will be generated immediately after a [Section 6.453](#) event in this case.

6.8 AddChildren Method 310

Applies To: [Section 6.629](#)

Description

This method is used to add child items to an item in a [Section 6.629](#) object

The argument to AddChildren is a 3, 4 or 5 element array as follows:

[1]	Item number	Integer.
[2]	New items	Vector of character vectors.
[3]	Depth vector	Integer vector.
[4]	Picture vector	Integer vector.
[5]	Selected picture vector	Integer vector.

Item number specifies the index of the item to which the child items are to be added.

New items is a vector of character vectors containing the labels for the new child items.

Depth vector is an integer vector specifying the depth of each of the new items relative to the parent item to which they are being added. The first element of this array must be 0.

Picture vector and *Selected picture vector* are optional and specify values of [Section 6.315](#) and [Section 6.518](#) respectively for each of the new items.

The result is the index at which the first new item has been inserted.

6.9 AddCol Event 153

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported by the [Section 6.263](#) object if the user presses the Cursor Right key, and the current cell ([Section 6.133](#)) is within the last column on the [Section 6.263](#). The default action is to append a new column to the contents of the [Section 6.263](#). If you attach a callback function to this event and have it return a value

of 0, a new column will not be appended to the [Section 6.263](#). Note that the event will not be generated unless the second element of the [Section 6.36](#) property is set to 1.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'AddCol ' or 153
[3]	Column number	number of the new column (integer)

An application may insert a new column into a [Section 6.263](#) by calling AddCol as a method. The argument is a 1 to 7-element array as follows:

[1]	Column number	number of the new column (integer)
[2]	Column title	character vector or matrix
[3]	Column width	integer
[4]	Undo flag	0 or 1
[5]	Resize flag	0 or 1
[6]	Title colour	negative integer or 3-element RGB vector
[7]	Line type	integer

If you are using default column headings, *Column title* will be ignored and the columns will be re-labelled with the default titles. If you have set [Section 6.118](#), the title you specify will be inserted. If you omit *Column title*, a blank title will be inserted.

Similarly, if you have not previously set [Section 6.84](#), [Section 6.481](#), [Section 6.117](#) or [Section 6.107](#), or if you have given them a scalar value, the corresponding parameter will be ignored. However, if you have specified [Section 6.84](#), [Section 6.481](#), [Section 6.117](#) or [Section 6.107](#) to be a *vector*, the number you specify in the corresponding parameter will be inserted into the appropriate property vector.

If you omit to specify *Column width* for the new column, it will be assigned a default value; new values for the other properties default to 0.

Undo flag (default 1) specifies whether or not the addition of the new column may subsequently be undone by an [Section 6.635](#) event.

To insert a new column before the first one, you must specify the *Column number* as 1 (or 0 if $\square I O$ is 0). To add a new column after the last one, you may specify any number

greater than the current number of columns. The data in the new column will be set to 0 if the [Section 6.642](#) property is numeric, or to an empty character vector otherwise.

6.10 AddComment Method 220

Applies To: [Section 6.263](#)

Description

This method is used to add a new comment to a [Section 6.263](#).

The argument to AddComment is a 3, 4 or 5 element array as follows:

[1]	Row	integer
[2]	Column	integer
[3]	Comment text	character array
[4]	Height in pixels	integer
[5]	Width in pixels	integer

For example, the following statement associates a comment with the cell at row 2, column 1; the text of the comment is "Hello", and the size of the comment window is 50 pixels (high) by 60 pixels (wide).

```
F.G.AddComment 2 1 'Hello' 50 60
```

Note that if you specify a row number of -1, the comment is added to the corresponding column *title*. Similarly, if you specify a column number of -1, the comment is added to the corresponding row *title*.

The height and width of the comment window, specified by the last 2 elements of the argument are both optional. If the cell already has an associated comment, the new comment replaces it.

You can use a dfn to add several comments in one statement; for example:

```
(1 2)(2 3){F.G.AddComment α,ω}'Hello' 'Goodbye'
```

Note that just before the comment is displayed, the [Section 6.263](#) generates a [Section 6.545](#) event which gives you the opportunity to (temporarily) change the text and/or window size of a comment dynamically.

The comment text specified by the 5th element of the argument to *Dyalog APL Language: Nq* must be a simple character scalar, vector, matrix or vector of vectors. Text specified by a simple character vector will be wrapped automatically if necessary. A matrix or vector of vectors may be used to explicitly specify multi-line text. If the array is a vector whose first element is an opening brace (`{`), the text is assumed to be in rich-text format (RTF) and is displayed accordingly. Note that there is no way for the user to scroll the text in the comment window and it is entirely your responsibility to ensure that the size of the window is appropriate for its contents.

6.11 AddItems Method 308

Applies To: [Section 6.629](#)

Description

This method is used to add items to a [Section 6.629](#) object

The argument to AddItems is a 3, 4 or 5-element array as follows:

[1]	Item number	Integer.
[2]	New items	Vector of character vectors.
[3]	Depth vector	Integer vector.
[4]	Picture vector	Integer vector.
[5]	Selected picture vector	Integer vector.

Item number specifies the index of the item to which the child items are to be added.

New items is a vector of character vectors containing the labels for the new child items.

Depth vector is an integer vector specifying the depth of each of the new items relative to the parent item to which they are being added. The first element of this array must be 0. This element may be omitted. If so, it is assumed to be all 0s.

Picture vector and *Selected picture vector* are optional and specify values of [Section 6.315](#) and [Section 6.518](#) respectively for each of the new items.

The new items are inserted with the first one being placed at the same level in the hierarchy as the item specified in element [1].

The result is an integer that reports the index position at which the first of the new items has been inserted.

6.12 AddRow Event 152

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported by the [Section 6.263](#) object if the user presses the Cursor Down key, and the current cell ([Section 6.133](#)) is within the last row on the [Section 6.263](#). The default action is to append a new row to the contents of the [Section 6.263](#). If you attach a callback function to this event and have it return a value of 0, a new row will not be appended to the [Section 6.263](#).

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3 element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'AddRow' or 152
[3]	Row number	number of the new row (integer)

An application may insert a new row into a [Section 6.263](#) by calling AddRow as a method. The argument is a 1 to 7-element array as follows:

[1]	Row number	integer
[2]	Row title	character vector or matrix
[3]	Row height	integer
[4]	Undo flag	0 or 1
[5]	Resize flag	0 or 1
[6]	Title colour	negative integer or 3-element RGB vector
[7]	Line type	integer

If you are using default row titles, *Row title* will be ignored and the rows will be re-labelled with default titles. If you *have* set [Section 6.500](#), the title you specify will be inserted. If you omit *Row title*, a blank title will be inserted.

Similarly, if you have not previously set [Section 6.77](#), [Section 6.483](#), [Section 6.499](#) or [Section 6.493](#), or if you have given them a scalar value, the corresponding parameter will be ignored. However, if you have specified [Section 6.77](#), [Section 6.483](#), [Section 6.499](#) or [Section 6.493](#) to be a vector, the number you specify in the corresponding

parameter will be inserted into the appropriate property vector. If you omit *Row height*, it will be assigned a default value; new values for the other properties default to 0.

Undo flag (default 1) specifies whether or not the addition of the new row may subsequently be undone by an [Section 6.635](#) event.

To insert a new row before the first one, you must specify the *Row number* as 1 (or 0 if $\square_{IO}$ is 0). To add a new row after the last one, you may specify any number greater than the current number of rows. The data in the new row will be set to 0 if the [Section 6.642](#) property is numeric, or to an empty character vector otherwise.

6.13 Align Property

Applies To

Description

For an [Section 6.21](#), the Align property may be 'None' or 'Centre' ('Center'). If Align is 'None', the [Section 6.21](#) window is automatically resized to fit the AVI being played. If Align is 'Centre', the AVI is centred in the [Section 6.21](#) window. If the window is too small, the AVI is clipped.

For a [Section 6.55](#), [Section 6.382](#), or [Section 6.384](#) the Align property may be 'None', 'Left' or 'Right'. If the [Section 6.55](#) Style is 'Radio' or 'Check' this property specifies the position of the text relative to the button symbol. The default is 'Right'. For a [Section 6.55](#) with Style 'Push', the value of Align is 'None'.

For a [Section 6.55](#) with Style 'Radio' or 'Check' that is created as a child of a Grid the value of the Align property may also be 'Centre' or 'Center'. Either of these values causes the symbol part of the [Section 6.55](#) (the circle or checkbox) to be centred within the corresponding Grid cell(s).

For a [Section 6.143](#), the Align property specifies the horizontal alignment of the drop-down Calendar which may be 'Left' (the default) or 'Right'. This applies only if the Style of the DateTimePicker is 'Combo'.

For a single-line [Section 6.185](#), Align specifies the vertical alignment of the text. It may be 'None' (the default), 'Centre' or 'Center'.

For a [Section 6.382](#), [Section 6.384](#), or [Section 6.575](#), Align 'Right' is used to position the object at the right end of its parent [Section 6.383](#) or [Section 6.574](#). 'None' is equivalent to 'Left' which is the default.

For objects of type [Section 6.127](#), [Section 6.566](#), [Section 6.508](#), [Section 6.574](#), [Section 6.582](#), [Section 6.621](#) and [Section 6.624](#), Align may be 'None', 'Top', 'Bottom', 'Left' or 'Right'. It specifies to which (if any) of the four sides of the parent the object is anchored and also the default position and size of the object. Specifying Align typically causes the [Section 6.32](#) property to be set to appropriate values as follows :

Align	Attach
'Top'	'Top' 'Left' 'Top' 'Right'
'Bottom'	'Bottom' 'Left' 'Bottom' 'Right'
'Left'	'Top' 'Left' 'Bottom' 'Left'
'Right'	'Top' 'Right' 'Bottom' 'Right'

These settings cause the object to remain at a fixed distance (in pixels) from the corresponding edge of the parent. Furthermore, the object will have a fixed height or width, but its length will stretch and shrink as the [Section 6.227](#) is resized.

Note that this does not apply to a [Section 6.585](#) for which the default value of Attach is 'None' 'None' 'None' 'None', regardless of the value of Align.

The default value of Align is 'Right' for a vertical [Section 6.508](#), 'Bottom' for a horizontal [Section 6.508](#), and 'Top' for a [Section 6.127](#), [Section 6.582](#), [Section 6.585](#), [Section 6.621](#) and [Section 6.624](#). Furthermore, unless [Section 6.452](#) and [Section 6.556](#) are specified explicitly, the object is placed along the corresponding edge of its parent.

For a [Section 6.508](#) object, Align also determines the direction of a [Section 6.508](#) object unless it is overridden by setting [Section 6.303](#) or [Section 6.646](#) directly. If neither [Section 6.303](#) or VScroll is defined and Align is 'Top' or 'Bottom', a horizontal scrollbar is provided. If neither [Section 6.303](#) or [Section 6.646](#) is defined and Align is 'None', 'Left' or 'Right', a vertical scrollbar is provided.

Note

The value of the Align property may **only** be assigned by *Dyalog APL Language*: Wc and may **not** be changed using *Dyalog APL Language*: Ws.

6.14 AlignChar Property

Applies To: [Section 6.263](#)

Description

The AlignChar property specifies a character on which the data displayed in a column of a Grid is to be aligned vertically. It is useful to align columns of numbers that are formatted by the [Section 6.229](#) property. AlignChar may be a scalar or singleton that applies to all columns of the Grid, or a vector with one element per column.

If the data in the column is left-justified, it is aligned using the first occurrence of the alignment character in each cell counting from the left. If the data is right-justified, it is aligned using the first occurrence of the alignment character from the right-hand end of the text.

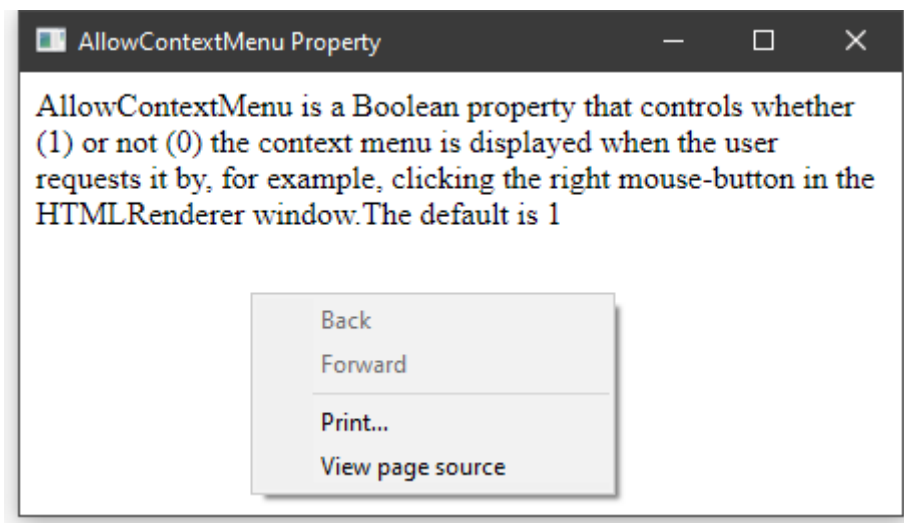
If the text in a cell does not contain an alignment character, it is aligned as if the alignment character were placed following the last digit.

6.15 AllowContextMenu Property

Applies To: [Section 6.307](#)

Description

This is a Boolean property that controls whether (1) or not (0) the context menu is displayed when the user requests it. The default is 1.



6.16 AlphaBlend Property

Applies To: [Section 6.227](#)

Description

The AlphaBlend property specifies a level of translucency which allows the area behind a Form to show through.

AlphaBlend is a scalar integer value in the range 0 to 255.

A value of 255 (the default) specifies no translucency, and the Form is entirely opaque obliterating anything behind it.

A value of 0 specifies total translucency and the Form itself is not visible. Furthermore, mouse events over the Form will not be reported by the Form itself but will be passed to any other windows underneath the Form.

Values in between specify varying levels of translucency.

6.17 AlwaysShowBorder Property

Applies To: [Section 6.263](#)

Description

The AlwaysShowBorder property specifies whether or not the border around the current cell in a [Section 6.263](#) is displayed when the [Section 6.263](#) loses the focus.

It is a Boolean value with a default value of 1.

6.18 AlwaysShowSelection Property

Applies To: [Section 6.263](#), [Section 6.348](#), [Section 6.629](#)

Description

The AlwaysShowSelection property specifies whether or not the selection remains highlighted when the object loses the focus.

It is a Boolean value with a default value of 1.

If AlwaysShowSelection is 1, the highlight is dimmed. If AlwaysShowSelection is 0, the highlight disappears.

6.19 AmbientChanged Event 533

Applies To: [Section 6.6](#), [Section 6.7](#)

Description

If enabled, this event is reported when any of the ambient properties change in an application hosting an [Section 6.7](#) object. The new values of the ambient properties are available from the [Section 6.225](#), [Section 6.205](#) and [Section 6.45](#) properties of the [Section 6.6](#).

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to -1 or by returning 0 from a callback function.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'AmbientChanged' or 533
[3]	Property code	integer
[4]	Description	character vector

For properties supported by Dyalog APL, Property code and Description may be one of the following:

Property code	Description	Meaning
-701	DISPID_AMBIENT_BACKCOLOR	BCol has changed
-703	DISPID_AMBIENT_FORECOLOR	FCol has changed
-705	DISPID_AMBIENT_FONT	Font has changed
-1	DISPID_AMBIENT_UNKNOWN	Unknown

Note that other ambient properties may be reported, although these have no corresponding Dyalog APL property.

6.20 Animate Method 29

Applies To

Description

The Animate method enables you to produce special effects when showing or hiding objects. There are three types of animation: roll, slide, and alpha-blended fade.

The argument to Animate is a 1 or 2-element array as follows:

[1]	Effects	integer
[2]	Play time	integer

The value of the *Effects* parameter is the sum of the following flags:

Flag	Value	Description
AW_HOR_POSITIVE	1	Animates the window from left to right. This flag can be used with roll or slide animation. It is ignored when used with the AW_CENTER flag.
AW_HOR_NEGATIVE	2	Animates the window from right to left. This flag can be used with roll or slide animation. It is ignored when used with the AW_CENTER flag.
AW_VER_POSITIVE	4	Animates the window from top to bottom. This flag can be used with roll or slide animation. It is ignored when used with the AW_CENTER flag.
AW_VER_NEGATIVE	8	Animates the window from bottom to top. This flag can be used with roll or slide animation. It is ignored when used with the AW_CENTER flag.
AW_CENTER	16	Makes the window appear to collapse inward if being hidden or expand outward if being displayed
AW_SLIDE	262144	Uses slide animation. By default, roll animation is used. This flag is meaningless on its own but is ignored when used with the AW_CENTER flag.
AW_BLEND	524288	Uses a fade effect. This flag can be used only for a Form.

The Playtime parameter is optional and specifies the length of time over which the animation is played in milliseconds. The default value depends upon the animation but is typically 200 milliseconds.

6.21 Animation Object

Purpose: The Animation object displays simple animations from basic .AVI files or resources.

Parents ActiveXControl Form Group PropertyPage SubForm

Children Bitmap Circle Cursor Ellipse Font Marker Poly Rect Text Timer

Properties Type Posn Size File Coord Border Active Visible Event Sizeable Dragable BCol AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Translate AcceptFiles KeepOnClose AutoPlay Transparent Align MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj AnimOpen AnimClose AnimPlay AnimStop

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap AnimStarted AnimStopped

Description

The Animation object displays simple animations from basic .AVI files or resources.

The Animation object can only play AVI files or resources that have no sound and can only display uncompressed AVI files or .AVI files that have been compressed using Run-Length Encoding (RLE).

For more sophisticated animations, you may use the Windows Media Player (OCX).

To display an AVI file, you must first use the [Section 6.23](#) method to open it. If the [Section 6.37](#) property is set to 1, the animation will play immediately. Otherwise, only the first frame will be displayed.

The [Section 6.13](#) property may be 'None' or 'Centre' ('Center'). If [Section 6.13](#) is 'None', the Animation window is automatically resized to fit the AVI being played. If [Section 6.13](#) is 'Centre', the AVI is centred in the Animation window. If the window is too small, the AVI is clipped.

The [Section 6.24](#) method may be used to play the animation and allows you to specify the start, number of frames, and repeat count.

The [Section 6.26](#) method causes the animation to stop.

The [Section 6.22](#) method closes the current AVI file and resets the contents of the object's window to its background colour.

The [Section 6.25](#) and [Section 6.27](#) events are reported when the animation starts and stops respectively.

6.22 AnimClose Method 291

Applies To: [Section 6.21](#)

Description

The AnimClose method closes the AVI file that is currently loaded in an Animation object. The display is reset to the object's background colour.

The AnimClose method is niladic.

6.23 AnimOpen Method 290

Applies To: [Section 6.21](#)

Description

The AnimOpen method opens an AVI file in an Animation object.

The argument to AnimOpen is a 1 or 2-element array as follows:

[1]	File	character vector
[2]	Resource id	integer

If a single element is specified, it represents the name of a .AVI file.

If 2 elements are specified, the first element specifies the name of a DLL or EXE and the second element identifies the particular AVI resource stored in that file. The identifier may be its name (a character string) or its resource id (a non-zero positive integer).

If the [Section 6.37](#) property is set to 1, the animation will play immediately. Otherwise, only the first frame will be displayed.

Note that the Animation object can only play AVI files or resources that have no sound and can only display uncompressed AVI files or .AVI files that have been compressed using Run-Length Encoding (RLE). If you attempt to open an inappropriate AVI file, the operation will fail with a DOMAIN ERROR and the following message will be displayed in the Status Window:

AVI file includes sound data or is in a format not supported by the Animation object

6.24 AnimPlay Method 292

Applies To: [Section 6.21](#)

Description

The AnimPlay method plays an AVI clip in an Animation object.

The argument to AnimPlay is a 3-element array as follows:

[1]	Repeat	integer
[2]	From	integer
[3]	To	integer

Repeat specifies the number of times the clip is repeated. A value of -1 causes the clip to be repeated indefinitely.

From is a 0-based index of the frame where playing begins and must be less than 65536. A value of zero means begin with the first frame in the AVI clip

To is a 0-based index of the frame where playing ends and must be less than 65536. A value of -1 means end with the last frame in the AVI clip

The last frame remains displayed until the clip is unloaded using [Section 6.22](#) or until another clip is loaded.

6.25 AnimStarted Event 294

Applies To: [Section 6.21](#)

Description

If enabled, this event is reported by an [Section 6.21](#) object just before an AVI clip starts playing

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'AnimStarted' or 294

This event is reported for information only and cannot be disabled or modified in any way.

6.26 AnimStop Method 293

Applies To: [Section 6.21](#)

Description

The AnimStop method stops playing an AVI clip in an Animation object.

AnimStop is niladic.

The last frame remains displayed until the clip is unloaded using [Section 6.22](#) or until another clip is loaded.

6.27 AnimStopped Event 295

Applies To: [Section 6.21](#)

Description

If enabled, this event is reported by an [Section 6.21](#) object just after an AVI clip has stopped playing

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'AnimStopped' or 295

This event is reported for information only and cannot be disabled or modified in any way.

6.28 APLVersion Property

Applies To: [Section 6.489](#)

Description

This is a read-only property that provides information about the Version of Dyalog APL that you are using. It is a 4-element vector of character vectors as described in the table below. In future releases these values may change, be removed, or new ones added.

Index	Description	Possible Values
[1]	Target Environment	Windows Windows-64 Windows Mobile Linux Linux-64 AIX AIX-64 Mac-64 Solaris Solaris-64
[2]	Version Number	
[3]	Version Type	W Windows S Server (terminal) version M Motif
[4]	Program Type	Development Runtime DLL DLLRT

EXAMPLE

'. 'WG'APLVersion'			
Windows-64	18.0.38434.0	W	Development

6.29 ArcMode Property

Applies To: [Section 6.94](#), [Section 6.190](#)

Description

This property determines how arcs are drawn. Its value is 0, 1 or 2.

0	only the arc is drawn
1	arcs define "arc segments", with a single straight line joining the two ends of the arc together
2	arcs define "pie segments", with lines drawn from the start and end points of the arc to the centre

Note that the segments defined by ArcMode 1 and 2 may be filled (by setting [Section 6.231](#)).

6.30 Array Property

Applies To: [Section 6.99](#)

Description

This property may be used to set or retrieve the contents of the Windows clipboard as a Dyalog APL array.

6.31 AsChild Property

Applies To: [Section 6.307](#)

Description

The AsChild property is a Boolean (default 0) indicating how the [Section 6.307](#) object is displayed. AsChild must be set when the object is created and may not subsequently be changed.

If AsChild is 0 (the default) the [Section 6.307](#) is displayed in a separate top-level window. If the [Section 6.307](#) is created (with AsChild 0) as a child of another object it still appears as a separate window and its Size and Posn relate to the screen rather than to its parent object. However, it is a member of its parent object's hierarchy and will disappear when its parent is closed.

If AsChild is 1, the [Section 6.307](#) must be created as a child of a valid parent type other than Root (which is not supported) and is displayed in a sub-window within its parent.

This property only applies to Microsoft Windows. On other platforms it is ignored.

6.32 Attach Property

Applies To

Description

This property specifies how an object responds to its parent being resized. It is a 4-element vector of character vectors which defines how each of the four edges of the object moves in response to a resize request made by the parent. Note that this property is only effective if the value of [Section 6.35](#) on the parent is 2 or 3 and [Section 6.35](#) for the object itself is 1 or 3.

The 4 elements of Attach refer to the Top, Left, Bottom and Right edges of the object respectively. Their values may be defined as follows :

Element	Value	Meaning
[1]	' T o p '	top edge of object attached to <i>top</i> edge of parent.
	' B o t t o m '	top edge of object attached to <i>bottom</i> edge of parent.
	' N o n e '	top edge of object is not attached to its parent
[2]	' L e f t '	left edge of object is attached to <i>left</i> edge of parent
	' R i g h t '	left edge of object is attached to <i>right</i> edge of parent
	' N o n e '	left edge of object is not attached to its parent
[3]	' T o p '	bottom edge of object is attached to <i>top</i> edge of parent.
	' B o t t o m '	bottom edge of object is attached to <i>bottom</i> edge of parent.
	' N o n e '	bottom edge of object is not attached to its parent
[4]	' L e f t '	right edge of object is attached to <i>left</i> edge of parent
	' R i g h t '	right edge of object is attached to <i>right</i> edge of its parent
	' N o n e '	right edge of object is not attached to its parent

If an edge of the object is attached to an edge of its parent, its position in absolute (pixel) terms remains fixed relative to that edge when its parent is resized. Thus if

Section 6.128 is 'Pixel', the corresponding Section 6.452 or Section 6.556 property of the object remains unaffected by the resize. If Section 6.128 has any other value, the value of Section 6.452 or Section 6.556 will change.

If an edge of the object is *not* attached to its parent, its absolute position (in pixels) will change in proportion to the size change (in the corresponding direction) of its parent. Thus if Section 6.128 is 'Pixel', the corresponding Section 6.452 or Section 6.556 property of the object will change as a result of the resize. If Section 6.128 has any other value, the value of Section 6.452 or Section 6.556 will be unaffected.

The default value of Attach is ('None' 'None' 'None' 'None'). This causes the object to reposition and resize itself in proportion to its parent.

Some objects have an Section 6.13 property which, among other things, provides a quick way to set their Attach property. Examining this mechanism may help to further explain how the Attach property works. Setting Section 6.13 to 'Top' has the effect of setting Attach to ('Top' 'Left' 'Top' 'Right'). Attaching the top edge of the object to the top edge of its parent causes the object to remain at a fixed distance from the top edge of its parent. The additional measure of attaching its bottom edge to the top edge of its parent causes the height of the object to remain fixed. Attaching the left and right edges of the object to the corresponding edges of its parent causes the object to shrink and expand as the parent is resized horizontally. If you position the object at (0 0) and set its width to be the same as the width of its parent, you have an object that always occupies the entire length of its parent, yet remains of fixed height. This is precisely the behaviour required for a Section 6.621 or a top Section 6.508 Bar. For further details, see Section 6.13 property.

6.33 AutoArrange Property

Applies To: Section 6.348

Description

The AutoArrange property is Boolean and specifies whether or not the items in a Section 6.348 object are automatically re-arranged when a single item is repositioned. Its default value is 0.

6.34 AutoBrowse Property

Applies To: Section 6.414

Description

This property is retained for backwards compatibility with previous versions of Dyalog APL, but is no longer relevant. Setting it has no effect.

6.35 AutoConf Property

Applies To

Description

This property determines what happens to an object when its **parent** is resized, and how resizing an object affects its children. It may take one of the following values; the default is 3.

0	Ignore resize by parent. Do not propagate resize to children.
1	Accept resize by parent. Do not propagate resize to children.
2	Ignore resize by parent. Propagate resize to children.
3	Accept resize by parent. Propagate resize to children.

If AutoConf is 0 or 2, the object's **physical** size (in pixels) and position (in pixels) relative to the top left corner of its parent remains unchanged when its parent is resized. If the object has 'Prop' or 'User' co-ordinates, the values of its [Section 6.452](#) and [Section 6.556](#) properties will change as a result.

If AutoConf is 1 or 3, by default, the object is physically reconfigured when its parent is resized such that its **relative** size and position within its parent remain unchanged. If the object has 'Pixel' co-ordinates, the values of its [Section 6.452](#) and [Section 6.556](#) properties will change as a result. Note that this default processing can be prevented by inhibiting the [Section 6.123](#) (31) Event.

If AutoConf is 0 or 1 and the object is resized, either by its parent or directly by the user, it does **not** attempt to physically reconfigure its children. This means that if the children have 'Prop' or 'User' co-ordinates, the values of their [Section 6.452](#) and [Section 6.556](#) co-ordinates will change as a result.

If AutoConf is 2 or 3 and the object is resized, either by its parent or directly by the user, it propagates a [Section 6.123](#) (31) Event to each of its children. By default this means that the object's children will be physically reconfigured so that they maintain their relative positions and sizes within it. If their co-ordinate system is 'Pixel', the values of their [Section 6.452](#) and [Section 6.556](#) properties will change as a result.

Note

Additional or alternative control may be imposed by inhibiting the [Section 6.123](#) (31) Event. This can be done either by setting the event's "action" code to `-1` or by returning a 0 from a callback function attached to it.

6.36 AutoExpand Property

Applies To: [Section 6.263](#)

Description

This property is a 2-element Boolean value that specifies whether or not rows and columns may be added to a [Section 6.263](#) object by the user.

If the first element of AutoExpand is 1, a row is added when the current cell is within the last row of the [Section 6.263](#) and the user presses Cursor Down

Similarly, if the second element is 1, a column is added when the current cell is within the last column of the [Section 6.263](#) and the user presses Cursor Right.

Note that when a row or column is added, the appropriate properties (including [Section 6.642](#) and [Section 6.82](#)) are expanded accordingly.

The default value for AutoExpand is (0 0).

If AutoExpand is enabled, the [Section 6.263](#) generates [Section 6.12](#) and [Section 6.9](#) events. You can return a zero from a callback function to selectively prevent the addition of rows and columns if appropriate.

6.37 AutoPlay Property

Applies To: [Section 6.21](#)

Description

Specifies whether or not an AVI clip is played immediately when loaded in an [Section 6.21](#) object.

AutoPlay is a single number with the value 0 (the default) or 1. If AutoPlay is 1, the AVI clip is automatically played through once from beginning to end when loaded from a file by the [Section 6.23](#) method.

6.38 BadValue Event 180

Applies To: [Section 6.56](#), [Section 6.185](#), [Section 6.563](#)

Description

If enabled, this event is reported by an [Section 6.185](#) or [Section 6.563](#) object when the user enters invalid data into the object and then switches focus to another control or to another application. Data is invalid if it conflicts with the [Section 6.206](#) property, or for a [Section 6.563](#) if it is outside the range specified by the [Section 6.345](#) property.

The default action of the event is to sound the bell (beep). You can disable this action by returning 0 from a callback function or by setting its action code to -1. Note that in neither case is the [Section 6.641](#) property of the object updated. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'BadVa lue ' or 180
[3]	Object name	character vector

The third element of the event message is either the name of the control to which the user has switched the focus, or is an empty vector if the focus has gone to another application.

6.39 BalloonHide Event 862

Applies To: [Section 6.581](#)

Description

If enabled, this event is reported by an [Section 6.581](#) object when a BalloonTip disappears for any reason other than it is dismissed by a timeout or because the user clicked the mouse.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'BalloonHide' or 862

This event is reported for information only and cannot be disabled or modified in any way

6.40 BalloonShow Event 861

Applies To: [Section 6.581](#)

Description

If enabled, this event is reported by an [Section 6.581](#) object when a BalloonTip is displayed using the [Section 6.543](#) method.

The event message reported as the result of `DDQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'BalloonShow' or 861

This event is reported for information only and cannot be disabled or modified in any way

6.41 BalloonTimeout Event 863

Applies To: [Section 6.581](#)

Description

If enabled, this event is reported by an [Section 6.581](#) object when a BalloonTip is dismissed by a timeout or because the user clicked the *close* (X) button..

The event message reported as the result of `DDQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'BalloonTimeout' or 863

This event is reported for information only and cannot be disabled or modified in any way

6.42 BalloonUserClick Event 864

Applies To: [Section 6.581](#)

Description

If enabled, this event is reported by an [Section 6.581](#) object when a BalloonTip is dismissed because the user clicked the mouse in the body of the BalloonTip. It is not reported when the user clicks the *close* (X) button.

The event message reported as the result of `OnBalloonUserClick`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'BalloonUserClick' or 864

This event is reported for information only and cannot be disabled or modified in any way

6.43 BandBorders Property

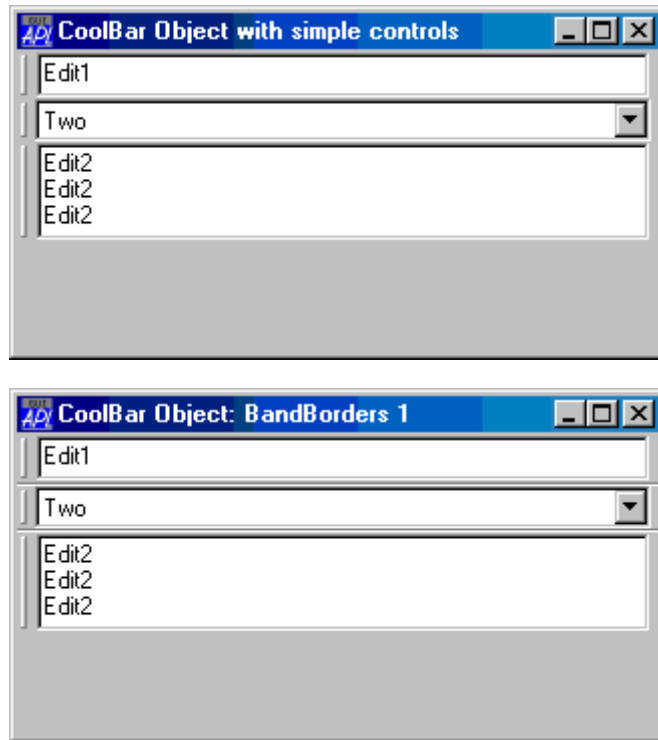
Applies To: [Section 6.127](#)

Description

The BandBorders property specifies whether or not narrow lines are drawn to separate adjacent bands in a [Section 6.127](#).

BandBorders is a single number with the value 0 (no lines) or 1 (lines are displayed); the default is 0.

The effect of BandBorders is illustrated below.



6.44 BaseClass Property

Applies To: [Section 6.408](#)

Description

This property specifies the name of the .NET class upon which the NetType is based. The newly created NetType inherits the properties, methods and events of this class.

6.45 BCol Property

Applies To

Description

This property defines the background colour(s) of an object.

For objects with more than one constituent part, it may specify a set of background colours, one for each part. A single colour is represented by a single number which

refers to a standard colour, or by a 3-element vector which defines a colour explicitly in terms of its red, green and blue intensities.

If BCol is set to 0 (which is the default) the background colour is defined by your current colour scheme for the object in question. For example, if you select yellow as your MS-Windows "Menu Bar" colour, you will by default get a yellow background in [Section 6.382](#) and [Section 6.384](#) objects, simply by not specifying BCol or by setting it to 0.

If BCol is set to 0 (Zilde), Dyalog APL will never paint the background of the object. If therefore the object is overlaid by another window and then exposed, its background will not be redrawn and it will simply contain whatever was previously shown on that area of the screen.

A negative value of BCol refers to a standard MS-Windows colour as described below. Positive values are reserved for a possible future extension.

BCol	Colour Element	BCol	Colour Element
0	Default	-11	Active Border
-1	Scroll Bars	-12	Inactive Border
-2	Desktop	-13	Application Workspace
-3	Active Title Bar	-14	Highlight
-4	Inactive Title Bar	-15	Highlighted Text
-5	Menu Bar	-16	Button Face
-6	Window Background	-17	Button Shadow
-7	Window Frame	-18	Disabled Text
-8	Menu Text	-19	Button Text
-9	Window Text	-20	Inactive Title Bar Text
-10	Active Title Bar Text	-21	Button Highlight

If BCol contains a 3-element vector, it specifies the intensity of the red, green and blue components of the colour as values in the range 0-255. For example, (255 0 0) is red and (255 255 0) is yellow.

Note that the colour realised depends upon the capabilities of the display adapter and driver, and the current Windows colour map.

For a [Section 6.55](#), BCol is only effective if the [Section 6.577](#) is 'Radio' or 'Check' and is ignored if the [Section 6.577](#) is 'Push'.

It is recommended that you only use **pure** background colours in [Section 6.121](#) and [Section 6.185](#) objects. This is because the text written in these objects cannot itself have a dithered background.

For the [Section 6.190](#), [Section 6.450](#) and [Section 6.472](#) objects, BCol specifies the background colour of the line drawn around the perimeter of the object and is effective only when a non-solid line ([Section 6.360](#) 1-4) is used. It also specifies the colour used to fill the spaces between hatch lines if a hatch fill ([Section 6.231](#) 1-6) is used.

6.46 BeginEditLabel Event 300

Applies To: [Section 6.348](#), [Section 6.629](#)

Description

If enabled, this event is reported when the user clicks on an item in a [Section 6.348](#) or [Section 6.629](#) object that has the focus, and signals the start of an edit operation. The default processing for the event is to display a pop-up edit box around the item and to permit the user to change its text.

You may disable the operation by setting the action code for the event to -1. You may prevent a particular item from being edited by returning 0 from a callback function. You may also initiate the edit operation by calling BeginEditLabel as a method.

The event message reported as the result of `□□□`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'BeginEditLabel' or 300
[3]	Item number	Integer. The index of the item.

6.47 Bitmap Object

Purpose: A graphical object used to represent a bitmap which may be used both to display a picture or as a pattern (brush) used to fill other objects.

Parents ActiveXControl Animation Button CoolBand Form Grid Group ImageList
ListView Menu MenuBar MenuItem NetType OLEServer Printer ProgressBar

PropertyPage PropertySheet RichEdit Root StatusBar SubForm TCPSocket ToolBar
ToolButton ToolControl TrackBar TreeView UpDown

Children Circle Ellipse Font Image Marker Metafile Poly Rect Text Timer

Properties Type File Bits CMap KeepBits Size Coord Event FontObj YRange XRange Data
TextSize Translate Accelerator KeepOnClose CBits MaskCol MethodList ChildList
EventList PropList

Methods Detach FileRead FileWrite MakePNG MakeGIF GetTextSize

Events Close Create Select

Description

A Bitmap may be created either from a file (.BMP, .GIF or .PNG) or from APL arrays. To create a Bitmap object using *Dyalog APL Language: Wc*, you can either specify the [Section 6.207](#) property **or** the [Section 6.68](#) property, **or** the [Section 6.48](#) and [Section 6.104](#) properties.

If you specify [Section 6.207](#), it should contain the name of a bitmap file from which the bitmap is to be read. If omitted, a .BMP file extension is added. You may also load a Bitmap from a DLL or from the DYALOG.EXE executable. See [Section 6.207](#) property for details.

If instead you want to create a Bitmap dynamically from APL variables, you may do so in one of two ways.

For a palette of up to 256 colours, you may specify the image using the [Section 6.48](#) and [Section 6.104](#) properties. The alternative is to use the [Section 6.68](#) property which works for any size of colour palette.

If [Section 6.367](#) is non-zero, it specifies the transparent colour for the Bitmap. Any pixels specified with the same colour will instead be displayed in whatever colour is underneath the Bitmap. This achieves similar behaviour to that of an Icon.

The [Section 6.337](#) property has the value 0 or 1, and controls how a Bitmap is saved in the workspace. A value of 0 (the default) means that the values of [Section 6.68](#), [Section 6.48](#) and [Section 6.104](#) are **not** kept in the workspace. If you request the values of [Section 6.68](#), [Section 6.48](#) or [Section 6.104](#) with *Dyalog APL Language: Wg*, they are obtained directly from the corresponding Windows bitmap resource. When the workspace is)LOADED, the Bitmap is recreated from the associated file defined by the value of the [Section 6.207](#) property. Note that if this file doesn't exist when the

workspace is)LOADED, the Bitmap is not created, but no error is generated. However, when you reference the object you will get a VALUE ERROR.

If [Section 6.337](#) is 1, the values of [Section 6.68](#), [Section 6.48](#) and [Section 6.104](#) are stored permanently in the workspace, and are used to rebuild the Bitmap when the workspace is)LOADED. In this case, the file name (if any) is ignored. Setting [Section 6.337](#) to 1 uses more workspace, but may be more convenient if you want to distribute applications.

The [Section 6.556](#) property allows you to query the size of a Bitmap without having to retrieve the [Section 6.68](#) or [Section 6.48](#) property and then take its "shape". This will be noticeably faster for a large Bitmap. If you set the [Section 6.556](#) property using *Dyalog APL Language: Ws* the Bitmap is scaled to the new size.

A useful feature of a Bitmap is that it can be the parent of any of the graphical objects. This allows you to create or edit a bitmap by drawing lines, circles, etc. in it.

The [Section 6.212](#) (90) and [Section 6.213](#) (91) methods allow you to dynamically manage bitmap files (.BMP). The expression:

```
bmname.FileWrite
```

causes the Bitmap called `bmname` to be written to the file specified by the current value of the [Section 6.207](#) property. The file is automatically written in standard bitmap format. Similarly, the expression:

```
bmname.FileRead
```

causes the Bitmap called `bmname` to be redefined from the bitmap file specified by the current value of the [Section 6.207](#) property.

The [Section 6.362](#) and [Section 6.363](#) methods may be used to convert the image represented by a Bitmap object into a GIF or PNG data stream, suitable for display in a web browser. The [Section 6.603](#) method may be used to transfer a Bitmap on a TCP/IP socket.

Using a bitmap is always a 2-stage process. First you create a Bitmap object with *Dyalog APL Language: Wc*. Then you use it by specifying its name as a property of another object.

The [Section 6.447](#) property specifies the name of a Bitmap to be displayed in an [Section 6.7](#), [Section 6.55](#), [Section 6.227](#), [Section 6.279](#), [Section 6.313](#), [Section 6.378](#), [Section 6.558](#), [Section 6.573](#), [Section 6.574](#), [Section 6.575](#), [Section 6.578](#), [Section 6.582](#), or [Section 6.621](#).

The [Section 6.53](#) property specifies three Bitmaps to be used to represent the 3 states of a [Section 6.55](#), [Section 6.382](#) or [Section 6.384](#).

The [Section 6.231](#) property specifies the name of a Bitmap to be used as a pattern to fill a [Section 6.450](#), [Section 6.190](#) or [Section 6.472](#) object.

6.48 Bits Property

Applies To: [Section 6.47](#), [Section 6.99](#), [Section 6.136](#), [Section 6.309](#)

Description

This property defines the pattern in a [Section 6.47](#), [Section 6.136](#), or [Section 6.309](#) object, or the pattern of a bitmap stored in the Windows clipboard.

For a [Section 6.47](#), [Section 6.99](#) or [Section 6.309](#), Bits is an integer matrix each of whose elements represents the colour of the corresponding pixel in the bitmap. The colours are specified as 0-origin indices into the [Section 6.104](#) property, which itself defines the complete set of different colours (the colour map) used by the object.

Please note that Bits and [Section 6.104](#) may **only** be used to represent an image with a colour palette of **256 colours or less**. If the colour palette is larger, the values of Bits and [Section 6.104](#) reported by `WMG` will be (0 0). For a high-colour image, use [Section 6.68](#) instead.

For a [Section 6.136](#), Bits is a Boolean matrix which specifies the shape of the cursor. For a [Section 6.136](#) and [Section 6.309](#), Bits is used in conjunction with the [Section 6.366](#) property.

See [Section 6.104](#) for further details.

6.49 Border Property

Applies To

Description

This property specifies whether or not an object is displayed with a border around it. It is a single number with the value 0 (no border) 1 (Border), or 2. The value 2 applies only to a [Section 6.227](#) and is used in combination with ('EdgeStyle' 'Dialog') to obtain standard dialog box appearance

For a [Section 6.227](#) or [Section 6.578](#), the value of the Border property is only relevant if [Section 6.557](#), [Section 6.398](#), [Section 6.580](#), [Section 6.369](#) and [Section 6.388](#) are all 0.

Note

The value of the Border property may only be assigned by *Dyalog APL Language: Wc* and may **not** be changed using *Dyalog APL Language: Ws*.

6.50 Browse Method 585

Applies To: [Section 6.412](#), [Section 6.414](#)

Description

The Browse method is retained for compatibility with previous versions of Dyalog APL, but is no longer relevant and has no effect.

6.51 BrowseBox Object

Purpose: The BrowseBox object allows the user to browse for and select a folder or other resource.

Parents ActiveXControl CoolBand Form Grid OLEServer PropertyPage PropertySheet Root StatusBar SubForm TCPSocket ToolBar ToolControl

Children Timer

Properties Type Caption BrowseFor Target StartIn HasEdit Event Data Translate KeepOnClose MethodList ChildList EventList PropList

Methods Detach Wait

Events Close Create FileBoxOK FileBoxCancel

Description

The BrowseBox object is a dialog box that allows the user to browse for and select a folder (directory) or other resource.

For full functionality as described here, the BrowseBox object requires the Windows Shell Library SHELL32.DLL Version 4.71 or higher. The BrowseBox object also supports the enhanced functionality provided by SHELL32.DLL Version 5 (Windows 2000) if present.

The [Section 6.52](#) property specifies the type of resource and may be 'Directory' (the default), 'File', 'Computer' or 'Printer'.

The [Section 6.569](#) property specifies the path name where browsing should start.

The [Section 6.286](#) property specifies whether or not the dialog box contains an edit field into which the user can type the name of the folder or other resource, rather than browsing for it. The default is 0.

A BrowseBox may only be used by the execution of a modal □□□. The action code for the FileBoxOK and FileBoxCancel events must be set to 1 so that the appropriate result is returned by the modal □□□.

After the user has pressed OK or Cancel, the [Section 6.591](#) property contains the name of the chosen folder or other resource.

EXAMPLE

```

▼ DIR←{START_DIR}GetDir CAPTION;BB;MSG
[1]  A Ask user for a Directory name
[2]  A CAPTION specifies Caption for dialog box
[3]  A START_IN (optional) specifies starting directory
[4]  A DIR is empty if user cancels
[5]  :With 'BB'□□WC'BrowseBox'
[6]      :If 2=□□NC'START_DIR'
[7]          StartIn←START_DIR
[8]      :Else
[9]          StartIn←''
[10]     :EndIf
[11]     onFileBoxOK←onFileBoxCancel←1
[12]     Caption←CAPTION
[13]     HasEdit←1
[14]     MSG←□□□□' '
[15]     :If 'FileBoxOK'≡2>MSG
[16]         DIR←Target A = 3>MSG
[17]     :Else
[18]         DIR←''
[19]     :EndIf
[20] :EndWith
▼

```

6.52 BrowseFor Property

Applies To: [Section 6.51](#)

Description

The BrowseFor property is a character vector that specifies the type of resource to be the target of an object.

BrowseFor may be 'Directory' (the default), 'File', 'Computer' or 'Printer'.

6.53 BtnPix Property

Applies To: [Section 6.55](#), [Section 6.382](#), [Section 6.384](#)

Description

This property is used to customise the appearance of a [Section 6.55](#), [Section 6.382](#) or [Section 6.384](#). It specifies the names of or refs to up to 3 [Section 6.47](#) objects to be used to display the object under different circumstances. In general, BtnPix is a 3-element vector of character vectors or refs. However, if it defines a single [Section 6.47](#), it may be a single ref, a simple character scalar or vector, or an enclosed character vector.

The first [Section 6.47](#) is displayed when the object is shown in its normal state. For a [Section 6.55](#), this is when its [Section 6.571](#) is 0. The second [Section 6.47](#) is used for a [Section 6.382](#) or [Section 6.384](#), when the object is selected (highlighted), or for a [Section 6.55](#) when its [Section 6.571](#) is 1. The third [Section 6.47](#) is used when the object is disabled by having its [Section 6.5](#) property set to 0.

For a [Section 6.55](#) with [Section 6.577](#) 'Push', this means that when the user clicks the [Section 6.55](#), its appearance switches from the first to the second [Section 6.47](#), and then back again. To maintain the standard 3-D appearance, the [Section 6.47](#)s should contain the correct shadow lines around their edges. For [Section 6.55](#)s with [Section 6.577](#) 'Radio' or 'Check', the [Section 6.55](#) will display one or other of the two [Section 6.47](#)s according to its current [Section 6.571](#).

For example, to have a [Section 6.55](#) that displays a "Tick" or a "Cross" according to its [Section 6.571](#):

```
'YES' □WC 'Bitmap' 'C:\WDYALOG\YES.BMP'
'NO' □WC 'Bitmap' 'C:\WDYALOG\NO.BMP'
'f1.r1' □WC 'Button'('Style' 'Check')
               ('BtnPix' 'YES' 'NO')
```

6.54 Btns Property

Applies To: [Section 6.399](#)

Description

The Btns property determines the set of buttons to be displayed in a [Section 6.399](#). It is a simple vector (one button) or a matrix with up to 3 rows, or a vector of up to 3 character vectors specifying the captions for up to 3 buttons. The buttons are arranged along the bottom of the dialog box in the order specified.

Under Windows, there are restrictions on these buttons. However the property has been designed more generally to be useful under different GUIs and perhaps later revisions of Windows.

Under Windows, the Btns property may specify one of **six** sets of buttons as follows.

- 'OK '
- 'OK ' 'CANCEL '
- 'RETRY' 'CANCEL '
- 'YES' 'NO '
- 'YES' 'NO' 'CANCEL '
- 'ABORT 'RETRY' 'IGNORE '

If any other combination is specified, *Dyalog APL Language: Wc* and *Dyalog APL Language: Ws* will report a DOMAIN ERROR. The names of the buttons are however case-insensitive, so the system will accept 'ok ', 'Ok ', 'oK ' or 'OK '.

If the Btns property is not specified, it assumes a default according to [Section 6.577](#) as follows :

Style	Btns
'Msg' or 'Info'	'OK '
'Warn' or 'Error'	'OK ' 'CANCEL '
'Query'	'YES' 'NO '

If [Section 6.577](#) is not specified, Btns defaults to 'OK '.

6.55 Button Object

Purpose: Allows the user to initiate an action or to select an option using a button.

Parents ActiveXControl CoolBand Form Grid Group PropertyPage SubForm ToolBar ToolControl

Children Bitmap Circle Cursor Ellipse Font Marker Poly Rect Text Timer

Properties Type Caption Posn Size Style Coord Align State Default Cancel Border Justify Active Visible Event Sizeable Draggable FontObj FCol BCol Picture BtnPix CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj ReadOnly Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex Elevated Note MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ChooseFont

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap FontOK FontCancel Select DropDown

Description

The type of button displayed is determined by the [Section 6.577](#) property which may take the value 'Push', 'Radio', 'Check', 'Toggle', 'Split' or 'CommandLink'. Under Windows, 'Toggle' and 'Check' are treated identically.

'Split' and 'CommandLink' apply only to Windows Vista and later and require [Section 8.6](#)

. Otherwise the use of these Styles will produce a Button with Style 'Push'.

Push buttons are used to generate actions. When the user "presses" a pushbutton, it generates a [Section 6.513](#) event (30). To cause an action, you simply associate the name of a callback function with this event for the Button in question.

Radio buttons and Check boxes are used to select options. They each have two states between which the user can toggle by clicking the mouse. When the Button (option) is selected, its [Section 6.571](#) property has the value 1; otherwise it is 0.

Only one of a group of Radio buttons which share the same parent can be set ([Section 6.571](#) is 1) at any one time. Radio buttons are therefore used for a set of choices that are **mutually exclusive**. Check boxes however, may be set together to signify a

combination of options. These are used for making choices which are not mutually exclusive.

Radio and Check buttons **also** generate [Section 6.513](#) events when their [Section 6.571](#) changes, and you can attach callback functions to these events to keep track of their settings. However, as Radio and Check buttons are not normally used to generate actions, it is perhaps easier to wait until the user signifies completion of the dialog box in some way, and then query the [Section 6.571](#) of the buttons using *Dyalog APL Language: Wg*. For example, if you have a set of Radio or Check buttons in a [Section 6.279](#) called `f1.options`, the following statements retrieve their settings.

```
OPTIONS ← (⊞WN 'f1.options') ⊞WG∘'State'
```

or

```
OPTIONS ← ⊞WG∘'State' ∘∘ ⊞WN 'f1.options'
```

The [Section 6.66](#) property determines the text displayed in the Button. Its default value is an empty vector. If the [Section 6.66](#) property contains one or more linefeed characters (⊞UCS 10) the text is centre-justified ([Section 6.577](#) 'Push' only) or top-left justified and automatically wraps on white-space characters (such as space and tab) to fit in the width provided.

If [Section 6.577](#) is 'Radio' or 'Check', the text may be aligned to the left or right of the button graphic using the [Section 6.13](#) property. Its default value is 'Right'.

The CommandLink button has an icon displayed to the left of its [Section 6.66](#). The appearance of the icon is controlled by the [Section 6.189](#) property. *Elevation* is a feature of *User Account Control* in Windows 7. In addition to the [Section 6.66](#), additional text may be defined by its [Section 6.411](#) property. If provided, this is displayed below the [Section 6.66](#).

The Split [Section 6.55](#) has a drop-down button, similar to that provided by a [Section 6.121](#) object.

If [Section 6.452](#) is omitted, the button is placed in the centre of its parent. If either element of [Section 6.452](#) is set to $\emptyset$, the button is centred along the corresponding axis.

If [Section 6.556](#) is not specified, the size of the button is determined by its [Section 6.66](#). If either element of [Section 6.556](#) is set to $\emptyset$ the corresponding dimension is determined by the height or width of its [Section 6.66](#). This does not apply to multi-line buttons whose dimensions should be specified explicitly. If [Section 6.66](#) is not specified, or is set to an empty vector, the value of [Section 6.556](#) is set to a default value.

Button colours can be specified using [Section 6.205](#) and [Section 6.45](#). However, pushbuttons ([Section 6.577](#) 'Push') **ignore** [Section 6.45](#) and instead use the standard Windows colour.

The [Section 6.447](#) property is used to display a bitmap on a pushbutton. This property is a 2-element array containing the name of a [Section 6.47](#) object and the "mode" in which it is to be displayed. The default mode (3) is the most useful, as it causes the [Section 6.47](#) to be **superimposed** on the centre of the Button. The surrounding edges of the Button (which gives it its 3-dimensional appearance and pushbutton behaviour) are unaffected. Note that if Picture is set on a Button whose Style is 'Radio' or 'Check', the Button assumes pushbutton appearance, although its 'Radio' / 'Check' behaviour is preserved.

An alternative is to use the [Section 6.53](#) property. This property specifies the names of up to 3 [Section 6.47](#) objects. The first [Section 6.47](#) is displayed when the [Section 6.571](#) of the Button is 0. The second is displayed when its [Section 6.571](#) is 1. The third is shown when the Button is inactive ([Section 6.5](#) 0). [Section 6.53](#) is more flexible than [Section 6.47](#), but if you want your Button to exhibit pushbutton behaviour, you must design your bitmap accordingly.

The [Section 6.470](#) property is Boolean and specifies whether or not the user may change the state of the Button. It applies only to [Section 6.577](#) 'Radio' and [Section 6.577](#) 'Check'.

The user can interact with the [Section 6.55](#) by clicking it, which generates a [Section 6.513](#) Event or (Style 'Split') the drop-down which generates a [Section 6.177](#) Event.

6.56 ButtonEdit Object

Purpose: Allows user to enter or edit data.

Parents ActiveXControl Form Group PropertyPage SubForm

Children Circle Ellipse Font ImageList Marker Poly Rect Text Timer

Properties Type Text Posn Size Style Coord Align Border Justify Active Visible Event ImageListObj SelText Sizeable Dragable FontObj FCol BCol CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj FieldType MaxLength Decimals Password ValidIfEmpty ReadOnly FormatString Changed Value Translate Accelerator AcceptFiles KeepOnClose Transparent ImageIndex Redraw TabIndex Cue ShowCueWhenFocused HasClearButton MethodList ChildList EventList PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap Select BadValue KeyError Change DropDown

Description

The ButtonEdit object combines a single-line input field with a customisable button. It provides the same user and programmer interfaces as an [Section 6.185](#) object (Style 'Single').

The appearance of the button, which is displayed to the right of the input field, is determined by the [Section 6.317](#) property. When clicked, the object generates a [Section 6.177](#) event. There is no default processing for this event; it is up to the programmer to take the appropriate action via a callback function.

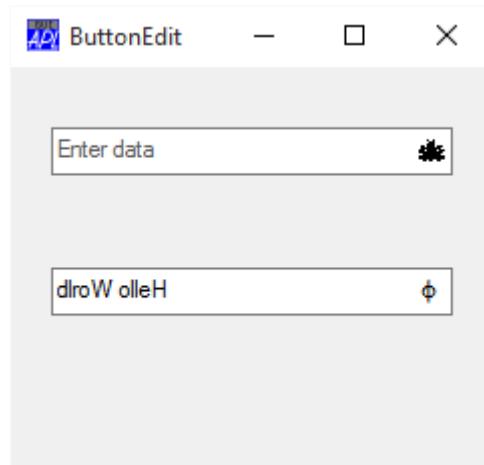
The following picture illustrates two ButtonEdit objects

```

▼ Example;BK;White;dialog
[1]  dialog←2 □NQ'.' 'GetEnvironment' 'Dialog'
[2]  'F'□WC'Form' 'ButtonEdit'('Coord' 'Pixel')('Size' 200 240)
[3]  'F'□WS'Coord' 'Pixel'
[4]  'F.IL1'□WC'ImageList'('Size' 16 16)('Masked' 1)
[5]  'F.IL1.Time'□WC'Icon'(dialog,'ws\arachnid.ico')
[6]  'F.BE1'□WC'ButtonEdit' ''(30 20)(Θ 200)
[7]  F.BE1.(Cue ShowCueWhenFocused)←'Enter data' 1
[8]  F.BE1.(ImageListObj ImageIndex)←F.IL1 1
[9]
[10] 'F.fnt'□WC'Font' 'APL385 Unicode' 16
[11] BK←16 16p256⊥White←255 255 255
[12] 'F.Rotate'□WC'Bitmap'('CBits'BK)('MaskCol'White)
[13] 'F.Rotate.'□WC'Text' 'φ'(0 3)('FontObj'F.fnt)
[14] BK←F.Rotate.CBits
[15] 'F.IL1.'□WC'Bitmap'('CBits'BK)('MaskCol'White)
[16] 'F.BE2'□WC'ButtonEdit' 'Hello World'(100 20)(Θ 200)
[17] F.BE2.(ImageListObj ImageIndex)←F.IL1 2
[18] F.BE2.onDropDown←'Rotate'
▼

▼ Rotate msg
[1]  (≡msg).Text←φ(≡msg).Text
▼

```



Note

For full functionality (in particular, for the [Section 6.132](#) property to apply), the ButtonEdit object requires that [Section 8.6](#) is enabled.

6.57 ButtonsAcceptFocus Property

Applies To: [Section 6.624](#)

Description

This is a Boolean property that determines how the Tab key and other cursor movement keys are handled by a [Section 6.624](#) object.

If ButtonsAcceptFocus is 0 (the default), when the user presses Tab or Shift+Tab to switch the input focus from another object to the [Section 6.624](#), the first [Section 6.623](#) in the [Section 6.624](#) receives the input focus and is highlighted. Pressing Tab or Shift+Tab again causes the input focus to move to another control. The cursor movement keys have no effect.

If ButtonsAcceptFocus is 1, when the user presses Tab or Shift+Tab to switch the input focus from another object to the [Section 6.624](#), the first or last [Section 6.623](#) in the [Section 6.624](#) receives the input focus and is highlighted. Note that the behaviour of Shift+Tab in this case is different. Pressing Tab or Shift+Tab again causes the input focus

to move to another control, although if there is no other control to accept the input focus, it moves to the first or last [Section 6.623](#) as appropriate. Pressing the cursor movement keys causes the input focus to move from one [Section 6.623](#) to the next.

6.58 Calendar Object

Purpose: The Calendar object provides an interface to the Month Calendar Control

Parents ActiveXControl Form Group PropertyPage SubForm ToolBar

Children Cursor Font Menu MsgBox TCPSocket Timer

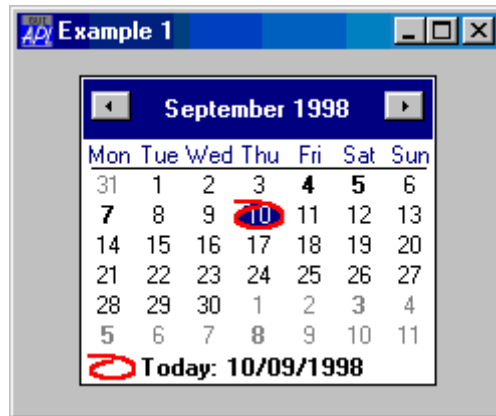
Properties Type Posn Size Style Coord Border Active Visible Event FirstDay MaxSelCount SelDate MinDate MaxDate CalendarCols Today HasToday CircleToday WeekNumbers MonthDelta Sizeable Dragable FontObj CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Accelerator KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj GetVisibleRange GetMinSize IDNToDate DateToIDN

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDblClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap SelDateChange GetDayStates CalendarDown CalendarUp CalendarDblClick CalendarMove Select

Description

The Calendar object displays a calendar and allows the user to select a date or range of dates. The following illustration shows a default Calendar object.



The Calendar object will display as many full months as it can fit into the area specified by its `Size` property as shown below. The minimum size required to encompass a single month may be obtained using the [Section 6.252](#) method.

```
'F'WC'Form' 'Example 2'('Size' 50 50)
'F.C'WC'Calendar'('Size' 100 100)
```



The [Section 6.620](#) property is an [Section 8.5](#) that specifies the current day. Its default value is today's date, that is, the local date set on your computer.

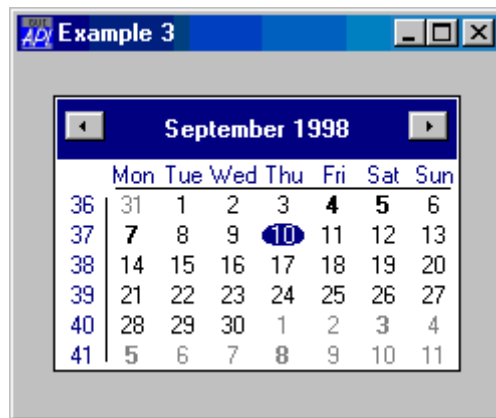
The [Section 6.95](#) property is either 0 or 1 (the default) and specifies whether or not the Today date is circled when the Calendar object is showing the corresponding month.

The [Section 6.290](#) property is either 0 or 1 (the default) and specifies whether or not the Today date is displayed (using the Windows short date format) in the bottom left of the Calendar object.

The [Section 6.656](#) property is either 0 (the default) or 1 and specifies whether or not the Calendar displays week numbers.

The following example shows a Calendar with both [Section 6.95](#) and [Section 6.290](#) set to 0 and [Section 6.656](#) set to 1.

```
'F'□WC'Form' 'Example 3'('Size' 30 30)
'F.C'□WC'Calendar'('CircleToday' 0)('HasToday' 0)('WeekNumbers' 1)
```



The [Section 6.217](#) property is an integer whose value is in the range 0-6. [Section 6.217](#) specifies the day that is considered to be the first day of the week and which appears first in the Calendar. The default value for [Section 6.217](#) depends upon your International Settings.

The [Section 6.389](#) and [Section 6.370](#) properties are integers that specify the minimum and maximum [Section 8.5](#) values that the user may display and select in the Calendar object. By default these properties specify the entire range of dates that the Windows Month Calendar control can provide.

The [Section 6.390](#) property specifies the number of months by which the Calendar object scrolls when the user clicks its scroll buttons. The default is empty (zilde) which implies the number of months currently shown.

The [Section 6.577](#) property may be either 'Single' (the default) or 'Multi'. If [Section 6.577](#) is 'Single', the user may select a single date. If [Section 6.577](#) is 'Multi', the user may select a contiguous range of dates. In this case, the maximum number of contiguous days that can be selected is defined by the [Section 6.372](#) property which is an integer whose default value is 7.

The [Section 6.511](#) property is a 2-element integer vector of [Section 8.5](#) values that identifies the first and last dates that are currently selected.

When the user selects one or more dates, the Calendar object generates a [Section 6.512](#) event. This event is also generated when the Calendar object is scrolled, and the selection changes automatically to another month.

The Calendar displays day numbers using either the normal or the bold font attribute and you may specify which attribute is to be used for each day shown. However, the Calendar object does not store this information beyond the month or months currently displayed.

When the Calendar control scrolls (and potentially at other times), it generates a [Section 6.243](#) event that, in effect, asks you (the APL program) to tell it which (if any) of the dates that are about to be shown should be displayed in bold.

If you wish *any* dates to be displayed using the bold font attribute, you must attach a callback function to the [Section 6.243](#) event which returns this information in its result. By default, all dates are displayed using the normal font attribute, so you only need a callback function if you want any dates to be displayed in bold.

You may also set the font attribute for particular days in the range currently displayed by calling [Section 6.243](#) as a method.

The [Section 6.59](#) property specifies the colours used for various elements in the Calendar object.

You may convert dates between [Section 8.5](#) and $\square$ TS representations using the [Section 6.312](#) and [Section 6.144](#) methods. Note that these methods apply to **all** objects and not just to the Calendar object itself.

The [Section 6.259](#) method reports the range of dates that is currently visible in the Calendar object.

6.59 CalendarCols Property

Applies To: [Section 6.58](#), [Section 6.143](#)

Description

The CalendarCols property specifies the colours used for various elements in the [Section 6.58](#) object.

CalendarCols is a 6-element integer vector whose elements specify the colours as follows:

[1]	Background colour displayed between months
[2]	Background colour displayed within the month.
[3]	Text colour within a month
[4]	Background colour displayed in the calendar's title
[5]	Colour used to display text within the calendar's title
[6]	Colour used to display header day and trailing day text. Header and trailing days are the days from the previous and following months that appear on the current month calendar.

Each element of CalendarCols may be 0 (which means default colour), a negative singleton that specifies a particular Windows colour, or a 3-element integer vector of RGB values.

Note: At the time of writing, setting the first element of CalendarCols has no effect. Dyalog believes this to be a Windows problem that may be corrected in due course.

6.60 CalendarDbClick Event 273

Applies To: [Section 6.58](#)

Description

If enabled, this event is reported when the user double-clicks the left mouse button over a [Section 6.58](#) object.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to `-1` or by returning 0 from a callback function.

The event message reported as the result of `□□□□□`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CalendarDbClick' or 273
[3]	Item Number	integer
[4]	Mouse Button	integer
[5]	Shift State	integer. Sum of 1=shift key, 2=ctrl key, 4=Alt key
[6]	Element Type	integer

For the meaning of elements 3 and 6, see [Section 6.61](#).

6.61 CalendarDown Event 271

Applies To: [Section 6.58](#)

Description

If enabled, this event is reported when the user depresses the left mouse button over a [Section 6.58](#) object.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to -1 or by returning 0 from a callback function.

The event message reported as the result of `DBQ`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CalendarDown' or 271
[3]	Item Number	integer (see below)
[4]	Mouse Button	integer
[5]	Shift State	integer. Sum of 1=shift key, 2=ctrl key, 4=Alt key
[6]	Element Type	integer (see below)

The 6th element of the event message is one of the following values:

If the value of the 6th element of the event message is 2 (CALENDARDATE), the 3rd element is the corresponding date reported as an [Section 8.5](#).

If the value of the 6th element of the event message is 5 (CALENDARDAY), the 3rd element is the index of the corresponding weekday (0-6).

If the value of the 6th element of the event message is 6 (CALENDARWEEKNUM), the 3rd element is the date of the first (leftmost) day in the corresponding week, reported as an [Section 8.5](#).

Otherwise, the 3rd element of the event message is 0.

6.62 CalendarMove Event 274

Applies To: [Section 6.58](#)

Description

If enabled, this event is reported when the user moves the left mouse button over a [Section 6.58](#) object.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to -1 or by returning 0 from a callback function.

The event message reported as the result of `□DQ`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CalendarMove' or 274
[3]	Item Number	integer
[4]	Mouse Button	integer
[5]	Shift State	integer. Sum of 1=shift key, 2=ctrl key, 4=Alt key
[6]	Element Type	integer

For the meaning of elements 3 and 6, see [Section 6.61](#).

6.63 CalendarUp Event 272

Applies To: [Section 6.58](#)

Description

If enabled, this event is reported when the user releases the left mouse button over a [Section 6.58](#) object.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to `-1` or by returning `0` from a callback function.

The event message reported as the result of `DDQ`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CalendarUp' or 272
[3]	Item Number	integer
[4]	Mouse Button	integer
[5]	Shift State	integer. Sum of 1=shift key, 2=ctrl key, 4=Alt key
[6]	Element Type	integer

For the meaning of elements 3 and 6, see Section 6.61.

6.64 Cancel Property

Applies To: Section 6.55

Description

This property determines which (if any) Push button in a [Section 6.227](#) or [Section 6.578](#) is to be associated with the Escape key. It has the value 1 or 0.

Pressing the Escape key will generate a [Section 6.513](#) event on the [Section 6.55](#) whose Cancel property is 1, regardless of which object has the keyboard focus.

As only one button in a [Section 6.227](#) or [Section 6.578](#) can be the Cancel button, setting Cancel to 1 for a particular button automatically sets Cancel to 0 for all others in the same [Section 6.227](#).

6.65 CancelToClose Method 367

Applies To: Section 6.463

Description

This method is used to change the buttons in a [Section 6.463](#) object. Its effect is to disable the Cancel button and, if the Style of the [Section 6.463](#) is 'Standard', it changes the text of the OK button to "Close". There is no result.

The CancelToClose method is niladic.

6.66 Caption Property

Applies To

Description

The Caption property is a character vector specifying fixed text associated with the object. For example, Caption defines the label on a [Section 6.55](#), the title of a [Section 6.227](#), [Section 6.578](#) or [Section 6.399](#), the heading in a [Section 6.279](#), and the text of a [Section 6.382](#) or a [Section 6.384](#).

For the [Section 6.489](#) object, Caption specifies the text displayed when Alt+Tab is used to switch to the Dyalog APL/W application. It may be used in conjunction with the [Section 6.310](#) property which specifies the name of an [Section 6.309](#) object to be displayed alongside this text.

Its default value is an empty vector.

For a [Section 6.55](#) or [Section 6.341](#), if the Caption property contains one or more linefeed characters (`␣`UCS 10) the text is top-left justified or, for a [Section 6.55](#) with [Section 6.577](#) 'Push', centre-justified; and automatically wraps on white-space characters (such as space and tab) to fit in the width provided.

For controls that support this feature, a single ampersand (&) is used to designate that the following character (if present) is an access key or an accelerator key and that character is underlined. The ampersand is not itself displayed. To negate this feature and cause an ampersand to be displayed, it is necessary to specify "&&".

6.67 CaseSensitive Property

Applies To: [Section 6.122](#)

Description

Specifies whether or not string searches in the items displayed by a ComboEx object will be case sensitive. Searching occurs when text is being typed into the edit box portion of the ComboEx

6.68 CBits Property

Applies To: [Section 6.47](#), [Section 6.99](#), [Section 6.309](#)

Description

The CBits property represents the pixels that make up a picture..

CBits provides an alternative representation to that provided by the [Section 6.48](#) and [Section 6.104](#) properties which apply only to images with 256 colours or under. CBits may be used to represent both low-colour and high-colour images.

CBits is a rank-2 numeric array whose dimensions represent the rows and columns of pixels in the image. The values in CBits represent the colour of each pixel and (for an [Section 6.309](#)) its transparency.

For a [Section 6.47](#), the colour value of each pixel is obtained by encoding the red, green and blue components, that is:

```
PIXEL ← 256 * RED + GREEN + BLUE
```

where RED, GREEN and BLUE are numbers in the range 0-255.

6.69 CEFVersion Property

Applies To: [Section 6.307](#)

Description

CEFVersion is a read-only property that reports the version of the [Chromium Embedded Framework \(CEF\)](#) that is being used. This information is primarily used for debugging and support.

It is a 10-element vector containing the following:

Index	Description
[1]	Formatted CEF release number. This is the primary identifier for a version of CEF.
[2]	CEF major version.
[3]	Commit number.
[4]	Chromium version number.
[5]	Chromium version number.
[6]	Chromium version number.
[7]	Chromium version number.
[8]	GIT Hashes
[9]	GIT Hashes
[10]	GIT Hashes

6.70 CellChange Event 150

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported when the user changes the contents of a cell in a [Section 6.263](#) object and then attempts to move to another cell or to another control outside the [Section 6.263](#).

The purpose of this event is to give the application the opportunity to perform additional validation before the update occurs (and to prevent it if necessary) or to update other cells in the [Section 6.263](#) as a result of the change.

The default action for the CellChange event is to update the appropriate element of the [Section 6.642](#) property with the new data. This action can be disabled by returning 0 from the attached callback function. Notice however, that the user is not prevented from moving away from the cell. If you are using this event to perform additional validation and you require the user to correct the data before moving away, you may force the user back to the cell in question using the [Section 6.78](#) method.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is an 8-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'CellChange' or 150
[3]	Cell row	integer
[4]	Cell column	integer
[5]	New data	number or character array
[6]	Object name	character vector (name of object to which the user has transferred focus)
[7]	New cell (row)	integer
[8]	New cell (column)	integer

If the user moves to another cell in the [Section 6.263](#), the 6th element of the event message is the name of the [Section 6.263](#) object and elements 7 and 8 specify the new cell address (□IO dependent).

If the user switches the input focus to another control or selects a [Section 6.384](#), the 6th element of the event message contains the name of that control or [Section 6.384](#). If the user switches to another application, the 6th element of the event message is an empty character vector. In all these cases, the 7th and 8th elements are 0.

The 5th element of the event message contains the data value that will be used to update the [Section 6.642](#) property. This will be numeric if the [Section 6.206](#) of the associated [Section 6.185](#) object is Numeric, LongNumeric, Date, LongDate or Time. Otherwise, it will be a character array.

An application can update an individual cell in the Grid under program control by calling CellChange as a method. If so, the *New object*, *New cell row* and *New cell column* parameters may be omitted.

6.71 CellChanged Event 164

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported after the user has changed the contents of a cell in a [Section 6.263](#) object and then moved to another cell or to another control outside the [Section 6.263](#). The purpose of this event is to give the application the opportunity to

perform calculations, and perhaps to update other cells in the [Section 6.263](#) as a result of the change.

Note that this event is reported **after** the change has taken place, and after the [Section 6.642](#) property has been updated. Furthermore, neither setting the event action code to `-1` nor returning 0 from a callback function has any effect. If you wish to *validate* the new data you should use the [Section 6.70](#) (150) event instead.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is an 5-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'CellChanged' or 164
[3]	Cell row	integer
[4]	Cell column	integer
[5]	New data	number or character array

The 5th element of the event message contains the data value that has been used to update the [Section 6.642](#) property. This will be numeric if the [Section 6.206](#) of the associated Edit object is Numeric, LongNumeric, Date, LongDate or Time. Otherwise, it will be a character array.

If you want to update an individual cell under program control, you may call [Section 6.70](#), but not CellChanged, as a method.

6.72 CellDbClick Event 163

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported when the user double-clicks a mouse button whilst over a cell in a [Section 6.263](#). The purpose of this event is to allow an application to enable some special action on double-click. This event may not be disabled.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 9 element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CellDbClick' or 163
[3]	Y	y-position of mouse (number)
[4]	X	x-position of mouse (number)
[5]	Button	button pressed (number) 1 = left button 2 = right button 4 = middle button
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down 4 = Alt key is down
[7]	Cell row	integer
[8]	Cell column	integer
[9]	Title index	integer

The y and x position of the mouse are reported relative to the top-left corner of the [Section 6.263](#).

The cell row and column are $\square$ IO dependent

If the user clicks over a row *title*, the value reported for the column is -1 , and the value reported for Title index is the index of that row title in [Section 6.500](#), or, if [Section 6.500](#) is not defined, the row number. Column titles are handled in a similar fashion.

6.73 CellDown Event 161

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported when the user presses a mouse button down whilst over a cell in a [Section 6.263](#). The purpose of this event is to allow an application to display a pop-up [Section 6.382](#) or a [Section 6.355](#) over a cell in a [Section 6.263](#) or to take some other special action.

The default action is to generate a [Section 6.78](#) event which will then position the user on the new cell. This action can be prevented by returning 0 from the callback function, in which case the normally ensuing [Section 6.78](#) event will not occur.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is an 9 element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CellDown' or 161
[3]	Y	y-position of mouse (number)
[4]	X	x-position of mouse (number)
[5]	Button	button pressed (number) 1 = left button 2 = right button 4 = middle button
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down 4 = Alt key is down
[7]	Cell row	integer
[8]	Cell column	integer
[9]	Title index	integer

The y and x position of the mouse are reported relative to the top-left corner of the [Section 6.263](#).

The cell row and column are $\square$ IO dependent

If the user clicks over a row *title*, the value reported for the column is -1 , and the value reported for Title index is the index of that row title in [Section 6.500](#), or, if [Section 6.500](#) is not defined, the row number. Column titles are handled in a similar fashion.

An application **can** position the user on a particular cell in a [Section 6.263](#) by calling CellDown as a method, but it is recommended that a [Section 6.78](#) event is used instead.

6.74 CellError Event 157

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported when the user inserts invalid data into the [Section 6.185](#) object associated with a cell in a [Section 6.263](#) object and then attempts to move

to another cell or to another control outside the [Section 6.263](#). It is also reported if the user selects a [Section 6.384](#).

The default action for the CellError event is to sound the bell (beep). This action can be disabled by returning 0 from the attached callback function. Whatever the result of the callback, the user will be prevented from moving to another cell in the [Section 6.263](#) and the [Section 6.133](#) and [Section 6.642](#) properties will remain unchanged. The user is not prevented from switching to any other control or to another application. However, if and when the user returns to the [Section 6.263](#), the current cell ([Section 6.133](#)) remains the invalid one and the user may not select a different one until the invalid data in the cell has been corrected. If you wish to allow the user to move to another cell without correcting the data, you may do so by generating a [Section 6.78](#) event explicitly. However, the [Section 6.642](#) property will remain unchanged and the invalid contents of the [Section 6.185](#) object will simply be discarded.

The event message reported as the result of *Dyalog APL Language: Dq* or supplied as the right argument to your callback function, is an 8-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'CellError' or 157
[3]	Cell row	integer
[4]	Cell column	integer
[5]	Invalid data	character vector
[6]	Object name	character vector (name of object to which the user has transferred focus)
[7]	New cell (row)	integer
[8]	New cell (column)	integer

If the user moves to another cell in the [Section 6.263](#), the 6th element of the event message is the name of the [Section 6.263](#) object and elements 7 and 8 specify the cell address (□IO dependent).

If the user switches the input focus to another control or selects a [Section 6.384](#), the 6th element of the event message contains the name of that control or [Section 6.384](#). If the user switches to another application, the 6th element of the event message is an empty character vector. In all these cases, the 7th and 8th elements are 0.

The 5th element of the event message contains the character vector in the [Section 6.605](#) property of the associated [Section 6.185](#) object which is inconsistent with its [Section 6.206](#).

6.75 CellFonts Property

Applies To: [Section 6.263](#)

Description

This property specifies the font objects to be used to display the [Section 6.642](#) in a [Section 6.263](#) object.

CellFonts is either a single ref or a simple character vector, or a vector of refs or character vectors. If it specifies a single font object this will be used to draw the text in *all* of the cells in the [Section 6.263](#). If it specifies more than one font object, these are mapped to individual cells through the [Section 6.82](#) property.

6.76 CellFromPoint Method 200

Applies To: [Section 6.263](#)

Description

This method converts from Grid co-ordinates to cell co-ordinates.

The argument to CellFromPoint is a 2-element array as follows:

[1]	y-coordinate	number in Grid co-ordinates
[2]	x-coordinate	number in Grid co-ordinates

The result is a 2-element vector containing the following:

[1]	y-coordinate	number in cell co-ordinates
[2]	x-coordinate	number in cell co-ordinates

6.77 CellHeights Property

Applies To: [Section 6.263](#)

Description

This property specifies the height of each row in a [Section 6.263](#) object in the units specified by its [Section 6.128](#) property. It may be a scalar or a vector whose length is the same as the number of rows implied by the [Section 6.642](#) property. If it is a scalar, it specifies a constant row height. If it is a vector it specifies the height of each row individually.

6.78 CellMove Event 151

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported when the user attempts to position the cursor over a cell in a [Section 6.263](#) by clicking the left mouse button or by pressing a cursor movement key. The purpose of this event is to allow an application to perform some action prior to the user entering a cell, to inhibit entry to a cell, or to deny exit from the current cell.

The default action is to position the user on the new cell. This action can be prevented by returning a 0 from the callback function attached to the event.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is an 8 element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CellMove' or 151
[3]	New cell row	integer
[4]	New cell column	integer
[5]	Scroll flag	0 or 1
[6]	Selection flag	0, 1 or 2
[7]	Mouse flag	0 or 1
[8]	Changed flag	0 or 1 (relates to current cell)
[9]	New value	new value of current cell or 0

The 5th element of the event message is 1 if switching to the new cell would cause the [Section 6.263](#) to scroll.

The 6th element of the event message is 1 if the user is moving to the new cell by extending the selection. It is 2 if the user selects an entire row or column (by clicking on a title), which moves the current cell to the first one in the selection.

The 7th element of the event message is 1 if the mouse is used to switch to a new cell.

The 8th element of the event message is 1 if the user is attempting to move to the new cell from another cell in the Grid having typed in (as if to alter) the current cell.

The 9th element of the event message is the intended new value in the current cell or $\emptyset$ (zilde) if *Changed flag* is 0.

The CellMove event may be used to validate and refuse changes as the user navigates between cells. See also [Section 6.70](#).

An application can position the user on a particular cell in a [Section 6.263](#) by calling CellMove as a method. If so, the argument need contain only the *New cell row* and *New cell column* parameters.

6.79 CellOver Event 160

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported when the user moves the mouse pointer whilst over a cell in a [Section 6.263](#).

There is no default action for this event.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is an 9 element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CellOver' or 160
[3]	Y	y-position of mouse (number)
[4]	X	x-position of mouse (number)
[5]	Button	button pressed (number) 1 = left button 2 = right button 4 = middle button
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down 4 = Alt key is down
[7]	Cell row	integer
[8]	Cell column	integer
[9]	Title index	integer

The y and x position of the mouse are reported relative to the top-left corner of the [Section 6.263](#).

The cell row and column are $\square$ IO dependent

If the user moves the mouse pointer over a row *title*, the value reported for the column is -1, and the value reported for Title index is the index of that row title in [Section 6.500](#), or, if [Section 6.500](#) is not defined, the row number. Column titles are handled in a similar fashion.

6.80 CellSelect Property

Applies To: [Section 6.263](#)

Description

The [Section 6.263](#) supports the selection of a contiguous block of cells by the user, using the mouse and/or the keyboard. The ability to select a range of cells is determined by the CellSelect property.

CellSelect may be a character vector or a vector of character vectors comprising the following:

'Rows '	User may select an entire row by clicking on a row title and may select multiple rows by dragging the mouse over contiguous row titles.
'MultiRows '	Same as 'Rows ', but user may additionally select several non-contiguous rows and blocks of rows using the Ctrl key.
'Columns '	User may select an entire column by clicking on a column title and may select multiple columns by dragging the mouse over contiguous column titles.
'MultiColumns '	Same as 'Columns ', but user may additionally select several non-contiguous columns and blocks of columns using the Ctrl key.
'Partial '	User may select any rectangular block of cells by either dragging the mouse or using Shift+cursor keys.
'MultiPartial '	Same as 'Partial ', but user may additionally select multiple rectangular blocks of cells using the Ctrl key.
'Whole '	User may select the entire <u>Grid</u> by clicking in the space to the left of the column titles and above the row titles.
'Any '	Same as ('Rows ' 'Columns ' 'Partial ' 'Whole '). This is the default.
'Multi '	Same as ('MultiRows ' 'MultiColumns ' 'MultiPartial ' 'Whole ').
'None '	User may not select any cells in the <u>Grid</u> .

For example, the following expression would allow the user to select only whole rows and columns:

```
gridname WS 'CellSelect' ('Rows' 'Columns')
```

Setting CellSelect to ('Rows' 'Columns' 'Whole' 'Partial') is equivalent to setting it to 'Any'.

When the user performs a selection, the [Section 6.263](#) generates a [Section 6.277](#) event.

The range of cells currently selected is given by the [Section 6.519](#) property. You can obtain the current selection by querying this property with *Dyalog APL Language: Wg* and you can set it with *Dyalog APL Language: Ws*.

Note that the user may delete the contents of the selected range, or cut and copy them to the clipboard by pressing Delete, Shift+Delete or Ctrl+Insert respectively. The user may also replace the current selection with the contents of the clipboard by pressing Shift+Insert. These operations generate [Section 6.268](#), [Section 6.267](#), [Section 6.265](#) and [Section 6.275](#) events which you may disable (by setting the event action code to -1 or to which you may attach a callback function).

If more than one block of cells has been selected, these operations are honoured only if the blocks begin and end on the same rows or begin and end on the same columns. If so, the data placed in the clipboard is the result of joining the blocks horizontally or vertically as appropriate.

You can also invoke these events as methods. This allows you to attach these actions to MenuItems and Buttons. For example, the following expression could be used to implement Cut as a MenuItem:

```
name WC 'MenuItem' 'Cu&t'('Event' 'Select' '#gridname.GridCut')
```

In addition to the ability to copy blocks of cells through the clipboard, the user may also drag a block of cells from one part of the [Section 6.263](#) to another.

When the user places the mouse pointer over any of the four edges of a selected block of cells, the cursor changes from a cross to an arrow pointer. The user may now drag the border of the selected block to a new location. If the Ctrl key is pressed at the same time, the contents of the selected cells are *copied* to the new location. If not, the operation is a *move* and the original block of cells is cleared (emptied). In either case, the contents of the original block replace the contents of the target block (marked by the dragging rectangle) and the target block becomes selected. You may only move or copy a *single* block of cells in this way.

These operations generate a [Section 6.269](#) event. You may prevent the user from moving and copying blocks of cells by disabling this event (by setting its event action code to -1) or you may control these operations selectively with a callback function. Note that although the operation of *inserting* cells (using Ctrl+Shift) has not been implemented, you may provide this facility yourself with the information provided by the event message.

You may also move or copy a block of cells (which need not necessarily be selected) under program control by calling [Section 6.269](#) event as a method.

6.81 CellSet Property

Applies To: [Section 6.263](#)

Description

This property identifies which cells in a [Section 6.263](#) are *set* (that is, have values) and which are empty. Its purpose is to allow large numeric matrices containing blank cells to be displayed and edited efficiently.

The CellSet property is a Boolean matrix with the same shape as the [Section 6.642](#) property. If an element of CellSet is 0, the cell is defined to be empty. Empty cells are displayed as blank and the cell contents by the [Section 6.642](#) property are ignored.

A more direct way to handle empty cells is to set the corresponding elements of [Section 6.642](#) to empty vectors. However, if [Section 6.642](#) is otherwise entirely numeric, this makes the array nested when it would otherwise be simple. For large numeric matrices, this penalty can be severe. For example, a 100x100 array of 2-byte integer values occupies about 20Kb of workspace. Setting one or more elements of the array to an empty vector increases its size to 200Kb. However, because it is Boolean, the size of the CellSet property for a 100x100 array is only 1.27Kb and represents a significant space saving.

Note that if the [Section 6.642](#) property contains text and is therefore nested anyway, the CellSet property is not helpful in conserving workspace, although it may still be useful to separate empty cells from real data.

You can dynamically change a single element of CellSet using the [Section 6.525](#) method.

6.82 CellTypes Property

Applies To: [Section 6.263](#)

Description

This property specifies the type of each cell in a [Section 6.263](#) object. It is a matrix whose elements are origin-1 indices into other property arrays ([Section 6.205](#), [Section 6.45](#), [Section 6.75](#) and [Section 6.321](#)).

For example, if CellTypes[1;1] is 3, the first cell in the [Section 6.263](#) is displayed using the foreground colour specified by the 3rd element of [Section 6.205](#), the background colour specified by the 3rd element of [Section 6.45](#), and so forth. Note however that scalar property arrays are extended if necessary. Therefore if you require 5 different foreground colours but only one background colour, [Section 6.45](#) need specify only a single colour.

You can dynamically change a single element of CellTypes using the [Section 6.526](#) method.

6.83 CellUp Event 162

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported when the user **releases** a mouse button down whilst over a cell in a [Section 6.263](#). This event is a companion to the [Section 6.73](#) event and could be used to hide a pop-up which was displayed in response to the [Section 6.73](#). The CellUp event performs no default action and may not be disabled.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 9 element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CellUp' or 162
[3]	Y	y-position of mouse (number)
[4]	X	x-position of mouse (number)
[5]	Button	button released (number) 1 = left button 2 = right button 4 = middle button
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down 4 = Alt key is down
[7]	Cell row	integer
[8]	Cell column	integer
[9]	Title index	integer

The y and x position of the mouse are reported relative to the top-left corner of the [Section 6.263](#).

The cell row and column are $\square$ IO dependent.

If the user clicks over a row *title*, the value reported for the column is -1 , and the value reported for Title index is the index of that row title in [Section 6.500](#), or, if [Section 6.500](#) is not defined, the row number. Column titles are handled in a similar fashion.

An application **can** position the user on a particular cell in a [Section 6.263](#) by calling [Section 6.73](#) event as a method, but it is recommended that a [Section 6.78](#) event is used instead.

6.84 CellWidths Property

Applies To: [Section 6.263](#)

Description

This property specifies the width of each column in a [Section 6.263](#) object in the units specified by its [Section 6.128](#) property. It may be a scalar or a vector whose length is the same as the number of columns implied by the [Section 6.642](#) property. If it is a scalar, it specifies a constant column width. If it is a vector it specifies the width of each column individually.

6.85 Change Event 36

Applies To: [Section 6.56](#), [Section 6.121](#), [Section 6.185](#), [Section 6.488](#), [Section 6.563](#)

Description

If enabled, this event is reported when the user alters the text in a [Section 6.121](#) or [Section 6.185](#) object (by typing). The event is not applicable for a [Section 6.121](#) with [Section 6.577](#) 'Drop' because this [Section 6.577](#) does not allow the user to alter data. The Change event is not reported repeatedly as the user edits the data. Instead, it is reported when the user indicates that he has finished with the field by :

1. clicking on another object, **or**
2. causing an event on another object (without altering the input focus) which will fire a callback function or cause *Dyalog APL Language: Dq* to terminate. This can occur if the user chooses a [Section 6.384](#), or fires a [Section 6.55](#) with the [Section 6.148](#) or [Section 6.64](#) property by pressing Enter or Esc, or selects an object using an accelerator key.

The purpose of the Change event is to allow the application to validate data which has been newly entered to the field, before proceeding with another action. It is for this

reason that the event is fired not just when the input focus changes, but also when the user takes some action that could cause the application to do something else.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' Change ' or 36
[3]	Object name	character vector (name of object that is to receive the focus or generate an event)

If the focus is transferred to an external application, the third element is an empty vector.

The default processing for this event is to allow the focus to change (if applicable) and to reset the internal flag that indicates that the data in the field has changed.

If you disable the event by setting the "action" code to `-1`, or inhibit it by returning 0 from a callback, the focus change (if applicable) is allowed to proceed, but the internal flag is not reset. If you wish to prevent the focus change you must explicitly reset the focus back onto the object that generated the event.

If the event was generated because the user switched to another application, and you return a 0 from your callback (because the data was not valid), the flag marking the [Section 6.121](#) or [Section 6.185](#) as having been changed remains set. If the user returns to your application by re-focusing on the same [Section 6.121](#) or [Section 6.185](#), nothing happens immediately, but because the field is marked as changed (the flag was not reset) you will get another Change event when he leaves it. However, if the user returns to your application in some other way, for example, by focusing on another object or by selecting a [Section 6.384](#), a second Change event will be generated immediately.

The following function illustrates how Change events can be processed. The `Check` function referred to in `line[4]` is assumed to return 1 if the data is valid and 0 if not.

```

[0]  R←VALIDATE Msg
[1]  A Validates field contents after Change event
[2]
[3]  A Normal exit (R←1) if data is valid
[4]    →(R←Check>Msg)/Exit
[5]
[6]  A R now 0, so field remains marked as "changed"
[7]
[8]  A If user has switched to another application,
[9]  A we need take no further action because we will
[10] A get a second Change event when he returns.
[11] →(''≡3>Msg)/Exit
[12]
[13] A Display error box (prepared earlier)
[14] 'ERR' □WS 'Text' 'Data is invalid'
[15] □DQ'ERR'
[16]
[17] A Restore focus to bad field
[18] □NQ(>Msg)40
[19]
[20] Exit:

```

6.86 Changed Property

Applies To: [Section 6.56](#), [Section 6.185](#), [Section 6.462](#), [Section 6.488](#), [Section 6.563](#)

Description

The Changed property, in conjunction with the Change event, provides the means to control the validation of an object after the user has finished interacting with it.

Initially, the value of the Changed property of an object is set to 0. When the user gives the focus to the object and causes either the Text or (in the case of a [Section 6.563](#)) the Thumb property to be altered, the Changed property is immediately set to 1. When the object loses the input focus and the value of the Changed property is 1, the object generates a Change event. The value of the Changed property is then determined as follows:

If there is no callback function attached to the Change event, or if the Change event is disabled, the Changed property is reset to 0.

If an attached callback returns no result or returns 1, the Change property is reset to 0.

If an attached callback function returns 0, the Changed property is not altered and remains set to 1. The object will therefore generate another Change event when the user next tries to leave it, even if the text and/or Thumb are not altered this time.

Note that the object generates a Change event when it loses the focus *only* if the value of the Changed property is 1 at the time.

6.87 CharFormat Property

Applies To: [Section 6.488](#)

Description

The CharFormat property describes or applies formatting to the currently selected text in a [Section 6.488](#) object. If the selection is empty, it reports or specifies the default character formatting for the object. It is a 5-element nested array structured as follows:

[1]	A vector of character vectors which describes the text attributes and is comprised of the following keywords: 'Autocolour' default colour (Windows text colour) 'Bold' bold text 'Italic' bold text 'Underline' underlined 'StrikeOut' line through text 'Protected' protected (read-only) text
[2]	A character vector that specifies the face name of the font used to draw the text
[3]	Character height in Twips .
[4]	Text colour. A single integer or an enclosed vector of 3 RGB values. The default is 0 which implies the standard Windows text colour.
[5]	Integer specifying the vertical offset of the character from the base line in Twips . This is used to specify superscript (positive offset) and subscript (negative offset) symbols. The default value is 0.

When you set the character format using `WC` or `WS` the following rules apply:

If you just want to set a single text attribute (element 1) you may specify a simple vector, for example (`WS 'CharFormat' 'Protected'`) is valid and will add the protected text attribute to the current set of text attributes.

To unset a text attribute (element 1) you must insert the tilde (~) character before the name of the attribute. For example, the expression (`¶WS 'CharFormat' '~Bold'`) will turn the bold text attribute off.

You need only specify the number of elements required, but you must insert proper values for the elements you wish to remain unaltered. However, you may use '' in the first element to leave the text attributes unchanged.

If there is no text selected, CharFormat specifies the *default* character format, that is, the format that will be used to draw the next (and subsequent) characters that the user enters. If there is text selected it specifies the format of the selected block of text. If the format is not strictly homogeneous, ¶WG may report the format of the first character in the selected block, or, if the block contains characters which use completely different fonts, the result of (`¶WG 'CharFormat'`) will be empty.

(`¶WS 'CharFormat' ...`) will set the format of the currently selected block of text. To set the format of an arbitrary block of text you must select it first using (`¶WS 'SelText' ...`).

6.88 CharSet Property

Applies To: Section 6.222

Description

CharSet is an integer that specifies the character encoding of the Section 6.222 object.

This property applies to the Classic Edition only. In the Unicode Edition, its value typically has no effect.

The following table illustrates some of the character set encodings supported by Windows. Note that this set may vary according to the edition of Windows that is installed.

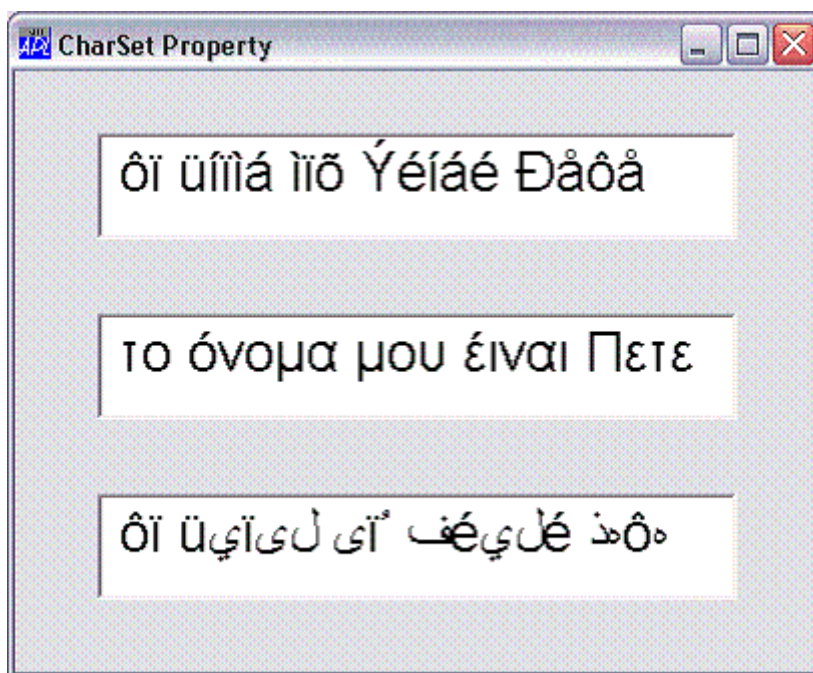
Language	CharSet
Western (ANSI)	0
Hebrew	177
Arabic	178
Greek	161
Turkish	162
Baltic	186
Central European	238
Cyrillic	204
Vietnamese	163

Windows fonts typically contain glyphs for the ASCII character set in their first 128 positions, and glyphs for the Western European character set in positions 129-256. Additional sets of character glyphs are stored in positions 257 onwards in what are sometimes referred to as codepages.

When you change the character set encoding, to (say) Greek (161), the set of Greek characters are mapped into the top 128 positions of the font.

For example, if the CharSet is 0 (ANSI), the character code Hex EC is displayed as ì (i-grave). However, if you change CharSet to 161 (Greek), the same character code is displayed as the Greek μ.

The following example illustrates how the character string 'ôï üíïía ìïö Ýéíáé ðåôå' appears differently according to the value of CharSet.



```
'F'WC'FORM' 'CharSet Property'('Size' 300 400)('Coord' 'Pixel')
F.Coord←'Prop'

'F.F1'WC'Font' ('Size' 32) ('CharSet' 0)
'F.F2'WC'Font' ('Size' 32) ('CharSet' 161)
'F.F3'WC'Font' ('Size' 32) ('CharSet' 178)

STRING←'ôï üíîá ïïö Ýéíáé Ðåôå'

'F.E1'WC'Edit'STRING(10 10)(0 80)('Font' 'F.F1')
'F.E2'WC'Edit'STRING(40 10)(0 80)('Font' 'F.F2')('Translate' 'ANS
I')
'F.E3'WC'Edit'STRING(70 10)(0 80)('Font' 'F.F3')('Translate' 'ANS
I')
```

Note

Setting Translate to 'ANSI' means that characters entered into the Edit fields using a corresponding National Language keyboard (Greek and Arabic) will appear in that language as intended.

6.89 CheckBoxes Property

Applies To: [Section 6.348](#), [Section 6.629](#)

Description

The CheckBoxes property specifies whether or not check boxes are displayed alongside items in a [Section 6.348](#) or [Section 6.629](#) object.

CheckBoxes is a single number with the value 0 (check boxes are not displayed) or 1 (check boxes are displayed); the default is 0.

For a [Section 6.629](#), CheckBoxes will only be honoured if the items have pictures associated with them (via the [Section 6.317](#) and [Section 6.315](#) properties).

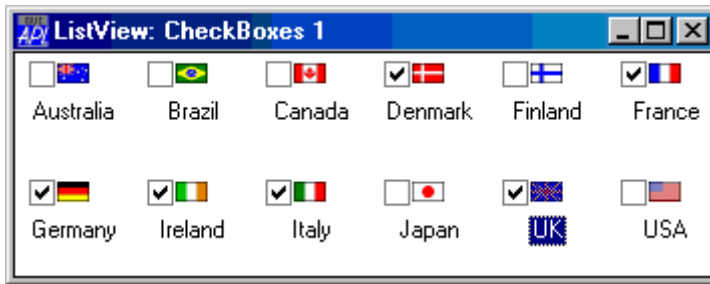
For a [Section 6.348](#), CheckBoxes applies to all settings of the [Section 6.644](#) property.

The GetItemState method can be used to determine if a specific item in a [Section 6.348](#) or [Section 6.629](#) is checked. The result of the method will have the 13th bit set if the item is checked.

```
STATE←Form.ListView.GetItemState 11
13>φ(32ρ2)τSTATE
```

1

The picture below illustrates the effect on the appearance of a [Section 6.348](#) object, of setting CheckBoxes to 1.



6.90 Checked Property

Applies To: [Section 6.384](#)

Description

This property determines whether or not a check mark is displayed alongside the text in a [Section 6.384](#). It is a single number with the value 0 (not checked) or 1 (checked). The default is 0.

6.91 ChildEdge Property

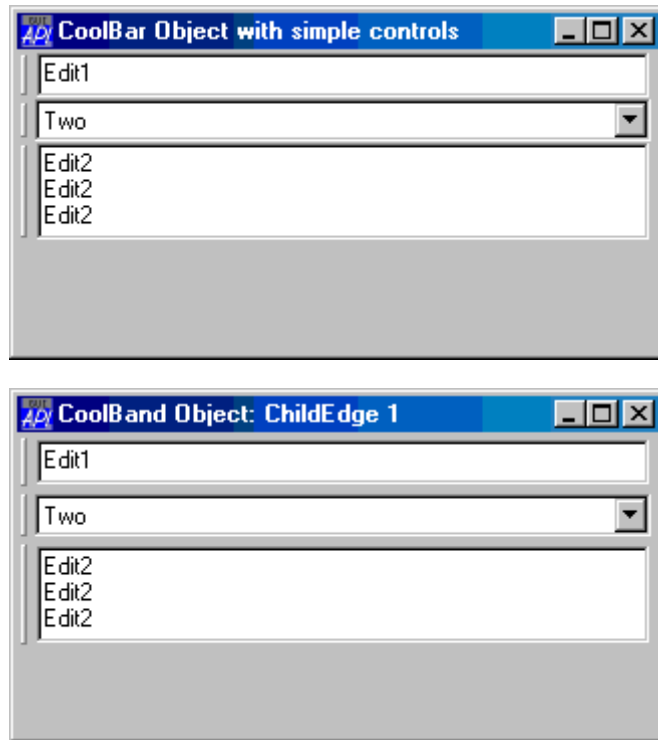
Applies To: [Section 6.126](#)

Description

The ChildEdge property specifies whether or not the [Section 6.126](#) leaves space above and below its child window.

ChildEdge is a single number with the value 0 (no space) or 1 (space is provided); the default is 0.

The following pictures illustrate the effect of the ChildEdge property.



6.92 ChildList Property

Applies To

Description

This is a read-only property that reports the names of those objects that may be created as children of the object in question. It is a vector of character vectors in which the order of the items is not significant.

6.93 ChooseFont Method 240

Applies To

Description

This method is used to display the standard Windows font selection dialog box.

The argument to ChooseFont is $\emptyset$ or a 1 or 2-element array as follows:

[1]	Printer name	character scalar or vector.
[2]	Modify flag	0 or 1.

If the argument is $\emptyset$ or the first element of the argument is ' ', the user is offered a list of fonts suitable for use on the screen. If not, the user is offered a choice of fonts suitable for the specified [Section 6.455](#) object. If you omit the 2nd element, the modify flag defaults to 0.

The dialog box is initialised with the properties of the [Section 6.222](#) object specified in the first element of the event message.

When the user presses the "OK" button, the "Cancel" button or closes the dialog box, ChooseFont terminates. Its result is either 0 (user pressed "Cancel") or a 2-element vector. In the latter case, the first element is an 8-element array that describes the selected font as described below, and the second element is a 3-element RGB colour vector.

[1]	Face name of selected font (character vector)
[2]	Character height in pixels (integer)
[3]	Fixed width or not (Boolean)
[4]	Italic or not (Boolean)
[5]	Underline or not (Boolean)
[6]	Weight (integer)
[7]	Angle of rotation (integer)
[8]	Character set (see CharSet)

If the modify flag was 1, the [Section 6.222](#) object is redefined to match the user's selections and all the objects that reference the [Section 6.222](#) are redrawn.

6.94 Circle Object

Purpose: A Graphical object to draw circles, arcs, and pie-slices.

Parents ActiveXControl Animation Bitmap Button ButtonEdit Combo ComboEx Edit Form Grid Group Label List ListView MDIClient Metafile Printer ProgressBar PropertyPage PropertySheet RichEdit Scroll Spinner Static StatusBar SubForm TabBar TipField ToolBar TrackBar TreeView UpDown

Children Timer

Properties Type Points Radius FCol BCol Start End ArcMode LStyle LWidth FStyle FillCol Coord Visible Event Dragable OnTop CursorObj AutoConf Data Accelerator KeepOnClose DrawMode RadiusMode MethodList ChildList EventList PropList

Methods Detach

Events Close Create DragDrop MouseDown MouseUp MouseMove MouseDbClick Help Select

Description

The [Section 6.449](#) property contains the co-ordinates of the centre of the circle. The size of the circle is determined by the [Section 6.467](#) property. This specifies the radius along the x-axis, the height is calculated so that the object is circular.

The [Section 6.468](#) property determines whether or not the circle is adjusted by a pixel, if required in order to appear perfectly round and perfectly centred. The default value is 0 (no adjustment is made).

The [Section 6.568](#) and/or [Section 6.192](#) properties are used to draw partial circles. They specify start and end angles respectively, measuring from the x-axis in a counter-clockwise direction and are expressed in radians. The type of arc is controlled by [Section 6.29](#) as follows :

ArcMode Effect

0	An arc is drawn from Start to End .
1	An arc is drawn from Start to End . In addition, a single straight line is drawn from one end of the arc to the other, resulting in a segment.
2	An arc is drawn from Start to End . In addition, two lines are drawn from each end of the arc to the centre, resulting in a pie-slice

.

[Section 6.449](#), [Section 6.467](#), [Section 6.568](#) and [Section 6.192](#) can specify vectors so that several arcs, circles, pie slices, etc. can be drawn in one call (and with one name).

If [Section 6.568](#) is specified, but not [Section 6.192](#), end angles default to (^1++ \start) , °2. If [Section 6.192](#) is specified, but not [Section 6.568](#), start angles default to 0, ^1++\End

This means that you can draw a pie-chart using either [Section 6.568](#) or [Section 6.192](#) angles; you do not have to specify both.

EXAMPLES

A circle whose centre is (50,50) and radius 20

```
'g.p1' WC 'Circle' (50 50) 20
```

An arc

```
'g.arc' WC 'Circle' (50 50) 20 ('Start' (00.75))('End' (01.25))
```

Complete pie

```
Data←12 27 21 40
ANGLES←0, -1÷((02)÷+/Data)×+\Data
COLS←(255 0 0)(0 255 0)(255 255 0)(0 0 255)
PATS←1 2 3 4
'g.pie' WC 'Circle' (50 50) 20 ('Start' ANGLES)
      ('ArcMode' 2) ('FCol' (0 0 0))
      ('FStyle' PATS) ('FillCol' COLS)
```

Same pie as above, but 2nd slice is exploded by changing its centre and 4th slice is shrunk by reducing its radius :

```
CY←50 52 50 50  A y-coord of centres
R←20 20 20 17.5 A radii
'g.pie' WC 'Circle' (50 CY) R ('Start' ANGLES)
      ('ArcMode' 1) ('FCol' (0 0 0))
      ('FStyle' PATS) ('FillCol' COLS)
```

6.95 CircleToday Property

Applies To: [Section 6.58](#), [Section 6.143](#)

Description

The CircleToday property specifies whether or not a circle is drawn around the [Section 6.620](#) date in a [Section 6.58](#) object or in the drop down calendar in a [Section 6.143](#), when the month containing that date is visible.

CircleToday is a single number with the value 0 (a circle is *not* drawn) or 1 (a circle is drawn); the default is 1.

See also [Section 6.290](#) property.

6.96 ClassID Property

Applies To: [Section 6.7](#), [Section 6.412](#), [Section 6.414](#), [Section 6.420](#)

Description

The ClassID property specifies the class identifier (usually abbreviated to CLSID) of an APL object that is used to represent a COM object. The CLSID is a globally unique identifier ([Section 8.4](#)) that uniquely identifies the object.

When you create or recreate an ActiveXControl or [Section 6.420](#) using `□WC`, you may specify ClassID. This allows you to re-use a value that was previously allocated to that control by the system. However, you should not specify any other value because that value could be allocated now or in the future to another object *on any other computer in the world*. Otherwise, a new ClassID is automatically allocated by the system.

Note that the CLSID is not actually recorded on your computer (in the registry) until you register it using `)SAVE` or `Make OCX`, or by executing the [Section 6.419](#) method.

6.97 ClassName Property

Applies To: [Section 6.7](#), [Section 6.412](#), [Section 6.414](#), [Section 6.420](#)

Description

For an [Section 6.414](#), the ClassName property specifies the name of the OLE object to which an [Section 6.414](#) object named by the left argument of `□WC` is to be connected. Similarly, for a [Section 2.41](#) the ClassName property specifies the name of the .NET class to be instantiated. Note that ClassName is mandatory for `□WC` and may not subsequently be changed using `□WS`.

For an [Section 6.7](#) or [Section 6.420](#), ClassName specifies the external name with which the object is registered, and by which it is referenced by other applications.

For an [Section 6.7](#), the external name is "Dyalog xxx Control, where xxx is the value of the ClassName property, or, if ClassName is not specified, the name of the [Section 6.7](#) namespace.

For an [Section 6.420](#), the external name is "Dyalog.xxx" where xxx is derived in the same way.

For a NetControl, the external name is the name of the .NET class which must be expressed relative to a corresponding element of `□USING`. For example, to load one of the standard .NET controls:

```
□USING, ←c 'System.Windows.Forms,system.windows.forms.dll'
```

6.98 ClickComment Method 225

Applies To: [Section 6.263](#)

Description

If enabled, a ClickComment event is generated when the user clicks the mouse in a comment window of a [Section 6.263](#).

The event message reported as the result of `□DQ`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ClickComment' or 225
[3]	Row	integer
[4]	Column	integer

The event message reports the co-ordinates of the cell. The default action is to raise the comment window so that it appears above all other, potentially overlapping, comment windows.

Note that if the comment window relates to a row or column *title*, the value reported in element [3] or [4] of the event message is `-1`.

6.99 Clipboard Object

Purpose: This object provides access to the Windows clipboard.

Parents ActiveXControl CoolBand Form OLEServer PropertyPage PropertySheet Root TCPSocket

Children Timer

Properties Type Event Data Formats Text Bits CMap CBits MetafileObj Picture Array RTFText Translate Accelerator KeepOnClose MethodList ChildList EventList PropList

Methods Detach Wait**Events** Close Create ClipChange Select**Description**

When an application places data in the Windows clipboard, it may store it in one or more formats. An application wishing to retrieve data from the clipboard can then choose which format to read it in. Dyalog APL supports standard clipboard formats, including CF_TEXT, CF_BITMAP and CF_METAFILE. If there is any data in the clipboard, the [Section 6.228](#) property lists the formats in which it may be retrieved.

In addition, the [Section 6.30](#) property may be used to set or retrieve clipboard contents in Dyalog APL array format.

Data is read from the clipboard using *Dyalog APL Language: Wg*, specifying the name of the appropriate property for the data that you want.

If the data has been stored in CF_Text format, the value of [Section 6.228](#) will include 'Text' and you may retrieve the data by querying the value of the [Section 6.606](#) property with *Dyalog APL Language: Wg*.

If the data has been stored in **device-independent** bitmap format, the value of [Section 6.228](#) will include 'CBits', 'Bits' and 'CMap'. To retrieve the bitmap pattern and colour map, you may query the values of the [Section 6.68](#), or [Section 6.48](#) and [Section 6.104](#) properties using *Dyalog APL Language: Wg*.

If the data has been stored in **device-dependent** bitmap format, only the bitmap pattern is available and [Section 6.228](#) will contain 'Bits' but not 'CMap'. In this case you can query the [Section 6.48](#) property but not [Section 6.104](#) without which you cannot realise the bitmap. However, if data was posted in this format, it is highly probable that the current Windows colour map applies to it. For a standard 16-colour device this is given under the description of the [Section 6.104](#) property.

The following example retrieves text from the clipboard :

```
'CL' ⍵WC 'Clipboard'
Data ← 'CL' ⍵WG 'Text'
```

The next example retrieves a bitmap from the clipboard and defines it as a [Section 6.47](#) object named 'BM' ready for use :

```
'BM' ⍵WC 'Bitmap' '', 'CL' ⍵WG 'Bits' 'CMap'
```

Data may be placed in the clipboard using *Dyalog APL Language: Wc* or *Dyalog APL Language: Ws*. To store text, you simply set the [Section 6.606](#) property. You may use a simple character vector or matrix, or a vector of character vectors. For example :

```
'CL' ⍴WS 'Text' 'Hello World'
```

To store a bitmap you can set either the [Section 6.447](#) property to the **name** of a [Section 6.47](#) object, or you can set the [Section 6.48](#) and [Section 6.104](#) properties explicitly. The former is more efficient, especially for large bitmaps, for example :

```
'CL' ⍴WS 'Bitmap' 'BM'
```

or

```
Bits CMap ← 'BM' ⍴WG 'Bits' 'CMap'
'CL' ⍴WS ('Bits' Bits)('CMap' CMap)
```

Note that if you use the latter method, you must set **both** properties in one *Dyalog APL Language: Ws* statement. This is also true if you wish to store data in both Text and Bitmap formats together.

The [Section 6.386](#) property allows graphical information to be restored in and retrieved from the clipboard in Windows Metafile format. See the description of the [Section 6.386](#) property for details.

A [Section 6.101](#) (120) event is generated when another application places data in the clipboard.

6.100 ClipCells Property

Applies To: [Section 6.263](#)

Description

This property determines whether or not the [Section 6.263](#) displays partial cells. The default is 1. If you set ClipCells to 0, the [Section 6.263](#) displays only complete cells and automatically fills the space between the last visible cell and the edge of the [Section 6.263](#) with the [Section 6.264](#) colour.

The first picture below shows a default [Section 6.263](#) (ClipCells is 1) in which the third column of data is in fact incomplete (clipped), although this is by no means apparent to the user. The second picture shows the effect on the [Section 6.263](#) of setting ClipCells to 0 which prevents such potential confusion.

ClipCells is 1 (the default)			
	A	B	C
1	13	75	4
2	51	83	
3	68	58	9
4	76	26	
5	98	72	7

ClipCells is 0			
	A	B	
1	13	75	
2	51	83	
3	68	58	
4	76	26	
5	98	72	

6.101 ClipChange Event 120

Applies To: [Section 6.99](#)

Description

If enabled, this event is reported when another application changes the contents of the Windows clipboard.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ClipChange' or 120

6.102 Close Event 33

Applies To

Description

A Close event is generated when an object is destroyed.

For a [Section 6.227](#) or [Section 6.578](#), the event may be generated by the user selecting "Close" from its System Menu. In this case, the event is reported **before** the window is destroyed, and you may prevent it from going ahead by associating a callback function which returns a result of 0.

By trapping this event you can control termination of your application in many different ways. For example, you could:

- automatically close all [Section 6.227](#) in your application when the master [Section 6.227](#) is closed.
- prevent the user from terminating the application if it is inappropriate at that time.
- display an "Are you sure ?" [Section 6.399](#).

Note that although all the object's child GUI objects will also be destroyed, none of the child objects will generate a Close event.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Close' or 33

6.103 CloseUp Event 46

Applies To: [Section 6.143](#)

Description

If enabled, this event is reported by a [Section 6.143](#) object just before the drop-down calendar is hidden. It applies only if the Style of the [Section 6.143](#) is 'Combo'.

The event message reported as the result of `□DQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'CloseUp' or 46

This event is reported for information only and cannot be disabled or modified in any way.

6.104 CMap Property

Applies To: [Section 6.47](#), [Section 6.99](#), [Section 6.136](#), [Section 6.309](#)

Description

This property defines the table of colours (the colour map) used by a [Section 6.47](#) or [Section 6.309](#) object or by a bitmap stored in the Windows clipboard. Its value is a 3-column integer matrix of numbers in the range 0-255. Each row represents a separate colour which is indexed (0-origin) by values in the [Section 6.48](#) property. The 3 columns refer to the intensities of the red, green and blue components of colour respectively.

Please note that [Section 6.48](#) and CMap may **only** be used to represent an image with a colour palette of **256 colours or less**. If the colour palette is larger, the values of [Section 6.48](#) and CMap reported by `WG` will be (0 0). For a high-colour image, use [Section 6.68](#) instead.

When you create a [Section 6.47](#) or [Section 6.309](#) by specifying [Section 6.48](#) and CMap, the actual colours you obtain are not necessarily those that you specified. This is partly due to hardware restrictions and partly due to the way in which Windows manages colours. Firstly, your display adapter and driver limit the number of pure colours that can be displayed at any one time and therefore define a maximum size for the colour map. For example, on a **standard** VGA you are limited to 16 different pure colours (additional ones are provided by **dithering**).

Secondly, Windows reserves a certain number of colours in the colour map for its own use. When an application requests a new colour (that is, one that is not already installed in the colour map), MS-Windows either assigns it to a spare entry, or allocates the **closest match** if the colour map is full. The value of [Section 6.48](#) and CMap after *Dyalog APL Language: Wc* reflect the actual colours allocated and may bear little resemblance to the values you assigned to these properties initially.

Note that if you are running 16 colours, MS-Windows reserves all 16 entries in the colour map for its own use. This means that on a 16-colour system, you **cannot** use any colours other than the default ones reserved by MS-Windows. In practice, the "standard" 16-colour CMap is shown in the following table.

Table: The default 16-colour CMap { #default-16-colour-cmap }

Bits[]	CMap			Colour
1	0	0	0	Black
2	128	0	0	Dark red
3	0	128	0	Dark Green
4	128	128	0	Olive Green
5	0	0	128	Dark Blue
6	128	0	128	Dark Magenta
7	0	128	128	Dark Cyan
8	128	128	128	Dark Grey
9	192	192	192	Light Grey
10	255	0	0	Red
11	0	255	0	Green
12	255	255	0	Yellow
13	0	0	255	Blue
14	255	0	255	Magenta
15	0	255	255	Cyan
16	255	255	255	White

If you are using a 256-colour set-up, the first 9 and the last 7 entries of the 256-colour CMap are the same as the first 9 and last 7 entries of the 16-colour CMap shown above. The intervening entries represent additional colours or are initially unused (0 0 0). New colours that you specify will be allocated to unused entries until the table is full.

6.105 ColChange Method 159

Applies To: [Section 6.263](#)

Description

This method is used to change the data in a column of a [Section 6.263](#).

The argument to ColChange is a 2-element array as follows:

[1]	Column number	integer
[2]	Column data	array

Note that the *Column data* must be a scalar or a vector whose length is equal to the number of rows in the [Section 6.263](#). Its elements may be scalar numbers, character vectors or matrices.

6.106 Collate Property

Applies To: [Section 6.455](#)

Description

Specifies whether or not multiple copies of printer output are collated.

Collate is a single number with the value 0 or 1. If Collate is 1, multiple copies of output are collated separately. If Collate is 0, copies are uncollated on output.

Collate is ignored unless [Section 6.129](#) is >1.

The default value for Collate is derived from the current printer setting and Collate is only effective if the printer supports this capability.

6.107 ColLineTypes Property

Applies To: [Section 6.263](#)

Description

This property specifies the appearance of the vertical grid lines in a [Section 6.263](#) object.

ColLineTypes is an integer vector, whose length is normally equal to the number of columns in the [Section 6.263](#). Each element in ColLineTypes specifies an index into the [Section 6.272](#) and [Section 6.274](#) properties, thus selecting the colour and width of the vertical grid lines.

For example, if ColLineTypes[1] is 3, the first vertical grid line in the [Section 6.263](#) is displayed using the colour specified by the 3rd element of [Section 6.272](#), and the width specified by the 3rd element of [Section 6.274](#).

Note that `ColLineTypes` is not `IO` dependent, and the value 0 is treated the same as the value 1; both selecting the *first* colour and line width specified by [Section 6.272](#) and [Section 6.274](#) respectively.

The default value of `ColLineTypes` is an empty numeric vector (`0`). If so, all vertical grid lines are drawn using the first element of [Section 6.272](#) and [Section 6.274](#).

A vertical grid line is drawn down the right edge of its associated column. One pixel is drawn *inside* the column of cells; additional pixels (if any) are drawn *between* that column of cells and the next one to its right.

6.108 ColorButton Object

Purpose: The `ColorButton` object allows the user to select a colour.

Parents `ActiveXControl` `Form` `Grid` `Group` `PropertyPage` `SubForm`

Properties `Type` `Caption` `Posn` `Size` `CurrentColor` `DefaultColors` `CustomColors` `OtherButton` `Coord` `Active` `Visible` `Event` `Sizeable` `Dragable` `CursorObj` `AutoConf` `Data` `Attach` `EdgeStyle` `Handle` `Hint` `HintObj` `Tip` `TipObj` `Translate` `Accelerator` `KeepOnClose` `ShowDropDown` `Redraw` `TabIndex` `MethodList` `ChildList` `EventList` `PropList`

Methods `Detach` `GetTextSize` `Animate` `GetFocus` `ShowSIP` `GetFocusObj`

Events `Close` `Create` `DragDrop` `Configure` `ContextMenu` `DropFiles` `DropObjects` `Expose` `Help` `KeyPress` `GotFocus` `LostFocus` `MouseDown` `MouseUp` `MouseMove` `MouseDownClick` `MouseEnter` `MouseLeave` `MouseWheel` `GestureZoom` `GesturePan` `GestureRotate` `GestureTwoFingerTap` `GesturePressAndTap` `ColorChange` `DropDown`

Description

The `ColorButton` object displays a coloured box, with an optional drop down button. When the user clicks the `ColorButton` with the left mouse button, a colour selection drop-down appears below it, allowing the user to select a new colour.

The [Section 6.134](#) property (default `0 0 0`) is a 3-element integer vector that specifies and reports the RGB value of the currently selected colour.

The [Section 6.149](#) property is a nested matrix which specifies the RGB values of the colours shown in the colour selection box. The shape of [Section 6.149](#) determines the number of rows and columns in the colour selection drop-down. Each element of [Section 6.149](#) is a 3-element integer vector specifying an RGB colour value.

The [Section 6.425](#) property is Boolean and specifies whether or not the user can select a colour using the Windows colour selection dialog box.

If [Section 6.425](#) is 1 (the default), the final row of the colour selection drop-down contains a button labelled "Other...". If the user clicks this button, the standard Windows colour selection dialog box is displayed, allowing the user to select any colour that the computer can render.

If [Section 6.425](#) is 0, the button labelled "Other..." is not present and the user is restricted to the choice of colours provided by the [Section 6.149](#) property.

The [Section 6.138](#) property is a 1-row, 16-column nested matrix which specifies the RGB values of the Colours displayed in the *Custom colors* section of the Windows colour selection dialog box. Each element of [Section 6.138](#) is a 3-element integer vector specifying an RGB colour value.

The [Section 6.548](#) property is Boolean (default 1) and specifies whether or not a drop-down button is displayed in the ColorButton object.

When the user clicks a ColorButton with the left mouse button, the object generates a [Section 6.177](#) event just before it displays the colour selection drop-down. This event may be used to set the [Section 6.149](#) and/or [Section 6.138](#) properties dynamically.

When the user selects a new colour, the ColorButton generates a [Section 6.109](#) event.

6.109 ColorChange Event 430

Applies To: [Section 6.108](#)

Description

If enabled, this event is reported by a [Section 6.108](#) object when the user chooses a colour from the colour selection drop-down.

The event message reported as the result of `[]DQ`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ColorChange' or 430
[3]	New Colour	3-element integer vector

The 3rd element of the event message contains the RGB value for the selected colour.

Note that the event is reported when the user chooses a colour, whether or not the newly selected colour differs from the one that was previously selected.

This event is reported for information only and cannot be disabled or modified in any way.

6.110 ColorMode Property

Applies To: [Section 6.455](#)

Description

Specifies whether or not printing is done in colour.

ColorMode is a single number with the value 0 or 1. If ColorMode is 1, printing is done in colour. If ColorMode is 0, printing is done using black ink only.

This property only applies to colour printers.

6.111 ColSorted Method 174

Applies To: [Section 6.263](#)

Description

This method is used to specify that an image is to be displayed in a [Section 6.263](#) column title to indicate the column has been sorted.

The argument to ColSorted is a 2-element array as follows:

[1]	Column number	Integer.
[2]	Sorted state	Integer. -1 = Sorted Down 0 = Unsorted 1 = Sorted Up

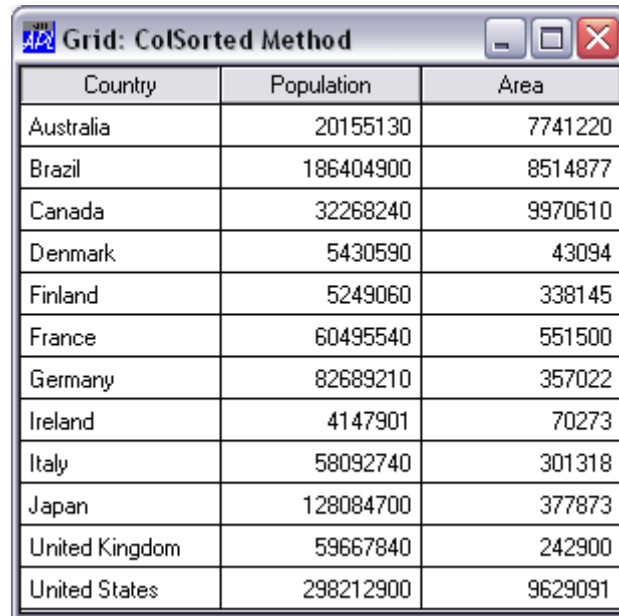
The column title for the appropriate column is redrawn to include the appropriate image.

If you wish to use your own images, you may specify them using the [Section 6.112](#) property.

```

'F'WC'Form' 'Grid: ColSorted Method'
'F.G'WC'Grid'('Posn' 0 0)(100 100)
F.G.Values←(COUNTRIES,POPULATION,[1.5]AREA)
F.G.ColTitles←'Country' 'Population' 'Area'
F.G.TitleWidth←0

```

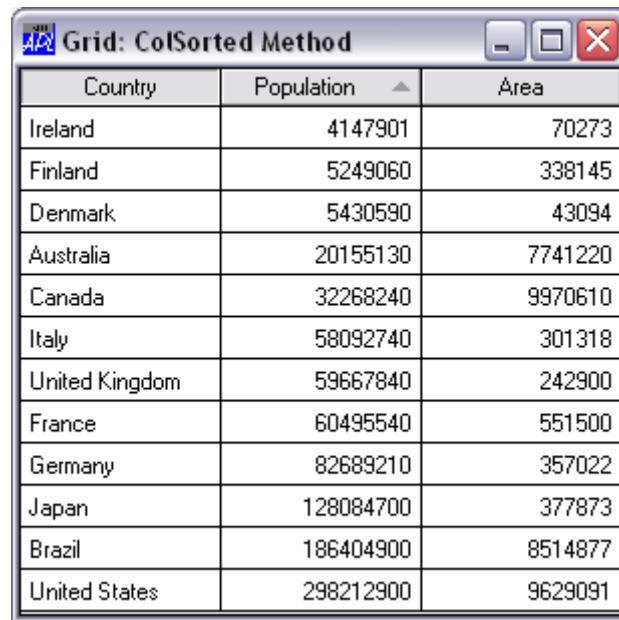


Country	Population	Area
Australia	20155130	7741220
Brazil	186404900	8514877
Canada	32268240	9970610
Denmark	5430590	43094
Finland	5249060	338145
France	60495540	551500
Germany	82689210	357022
Ireland	4147901	70273
Italy	58092740	301318
Japan	128084700	377873
United Kingdom	59667840	242900
United States	298212900	9629091

```

F.G.Values←(Values[⌵Values[:2];])
F.G.ColSorted 2 1

```

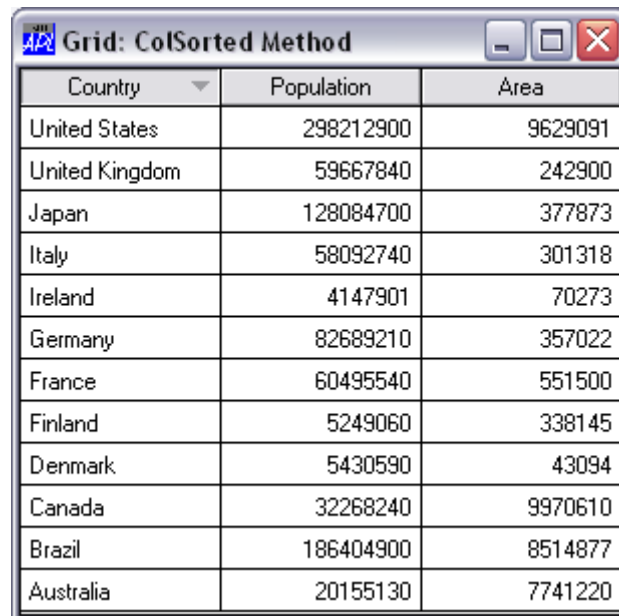


Country	Population	Area
Ireland	4147901	70273
Finland	5249060	338145
Denmark	5430590	43094
Australia	20155130	7741220
Canada	32268240	9970610
Italy	58092740	301318
United Kingdom	59667840	242900
France	60495540	551500
Germany	82689210	357022
Japan	128084700	377873
Brazil	186404900	8514877
United States	298212900	9629091

```
F.G.(Values←Values[↑↑Values[:,1];])
```

```
F.G.ColSorted 2 0
```

```
F.G.ColSorted 1 -1
```



Country	Population	Area
United States	298212900	9629091
United Kingdom	59667840	242900
Japan	128084700	377873
Italy	58092740	301318
Ireland	4147901	70273
Germany	82689210	357022
France	60495540	551500
Finland	5249060	338145
Denmark	5430590	43094
Canada	32268240	9970610
Brazil	186404900	8514877
Australia	20155130	7741220

6.112 ColSortImages Property

Applies To: [Section 6.263](#)

Description

The ColSortImages property identifies the names of, or refs to, up to three [Section 6.47](#) objects that are used to specify the sort images for a [Section 6.263](#) object.

If ColSortImages is not specified, default images are used.

The [Section 6.47](#) specified by the 1st element of ColSortImages is used to display columns that are sorted down.

The [Section 6.47](#) specified by the 2nd element of ColSortImages is used to display columns that are unsorted.

The [Section 6.47](#) specified by the 3rd element of ColSortImages is used to display columns that are sorted up.

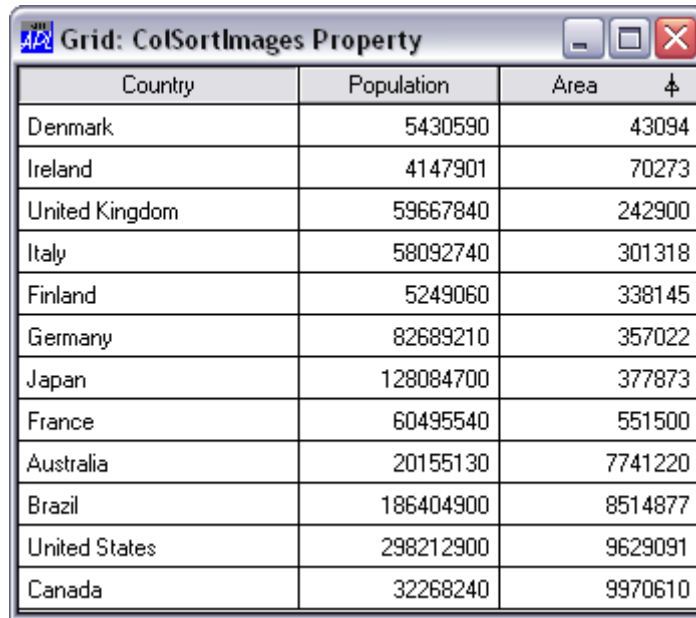
```
'F'□WC'Form' 'Grid: ColSortImages Property'
F.(Coord Size)←'Pixel'(313 341)
'F.fnt'□WC'Font' 'APL385 Unicode' 16
F.FontObj←F.fnt
BK←16 16p256↓White←255 255 255

'F.gu'□WC'Bitmap'('CBits'BK)('MaskCol'White)
'F.gu.'□WC'Text' '▲'(0 3)
'F.gd'□WC'Bitmap'('CBits'BK)('MaskCol'White)
'F.gd.'□WC'Text' '▼'(0 3)

'F.G'□WC'Grid'('Posn' 0 0)F.Size
F.G.Values←#.(COUNTRIES,POPULATION,[1.5]AREA)
F.G.ColTitles←'Country' 'Population' 'Area'
F.G.CellWidths←140 100 100
F.G.TitleWidth←0

F.G.ColSortImages←'F.gd' '' 'F.gu'
F.G.(Values←Values[▲Values[;3];])

F.G.ColSorted 3 1
```



Country	Population	Area
Denmark	5430590	43094
Ireland	4147901	70273
United Kingdom	59667840	242900
Italy	58092740	301318
Finland	5249060	338145
Germany	82689210	357022
Japan	128084700	377873
France	60495540	551500
Australia	20155130	7741220
Brazil	186404900	8514877
United States	298212900	9629091
Canada	32268240	9970610

6.113 ColTitle3D Property

Applies To: [Section 6.348](#)

Description

The ColTitle3D property is a Boolean value that specifies whether or not the column titles in a [Section 6.348](#) object are displayed with a 3-dimensional effect. Its default value is 1. A column heading with a 3-dimensional button appearance may be used to imply that the user may click on it to sort by the values in that column.

ColTitle3D is only relevant if [Section 6.644](#) is 'Report' and [Section 6.291](#) is 1. Note that this property may only be set by `WC` and may not subsequently be changed using `WS`.

6.114 ColTitleAlign Property

Applies To: [Section 6.263](#), [Section 6.348](#)

Description

The ColTitleAlign property specifies the alignment of column titles. For a [Section 6.348](#) object this is only relevant only when the [Section 6.644](#) property is set to 'Report'.

ColTitleAlign is either a simple character vector, or a vector of character vectors with one element per column.

For a [Section 6.263](#), ColTitleAlign may be: 'Top', 'Bottom', 'Left', 'Right', 'Centre', 'TopLeft', 'TopRight', 'BottomLeft', or 'BottomRight'.

For a [Section 6.348](#) object, ColTitleAlign may be 'Left', 'Right' or 'Centre'. Also, for a [Section 6.348](#) the column data itself is aligned likewise. Note that the *first* column in a [Section 6.348](#) is always left-aligned regardless of the setting of ColTitleAlign. This is a Windows restriction.

Note that both spellings 'Centre' and 'Center' are accepted.

6.115 ColTitleBCol Property

Applies To: [Section 6.263](#)

Description

The ColTitleBCol property specifies the background colour of the column titles in a [Section 6.263](#) object

ColTitleBCol may be a scalar that specifies a single background colour to be used for all of the column titles, or a vector that specifies the background colour of each of the column titles individually. An element of ColTitleBCol may be an enclosed 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or it may be a scalar that defines a standard Windows colour element (see [Section 6.45](#) for details). Its default value is 0 which obtains the colour defined for Button Face.

6.116 ColTitleDepth Property

Applies To: [Section 6.263](#)

Description

ColTitleDepth specifies the structure of a set of hierarchical column titles. It is an integer vector with the same length as the [Section 6.118](#) property. A value of 0 indicates that the corresponding element of [Section 6.118](#) is a top-level title. A value of 1 indicates that the corresponding title is a sub-title of the most recent title whose ColTitleDepth is 0; a value of 2 indicates that the corresponding title is a sub-title of the most recent title whose ColTitleDepth is 1, and so forth. For example:

```

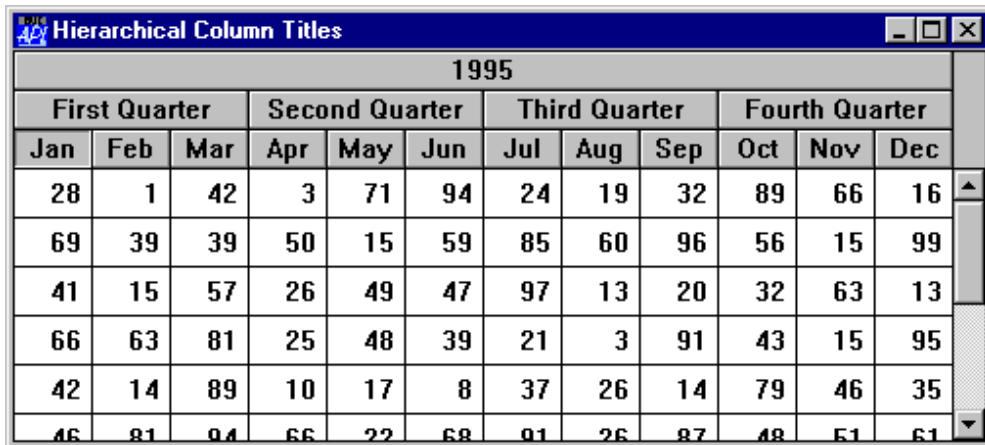
'F'WC'Form'('Coord' 'Pixel')('Size' 200 498)
'F'WS'Caption' 'Hierarchical Column Titles'
'F.G'WC'Grid'(?10 12p100)(0 0)(200 498)
'F.G'WS('TitleWidth' 0)('TitleHeight' 60)
'F.G'WS'CellWidths' 40

Q1←'First Quarter' 'Jan' 'Feb' 'Mar'
Q2←'Second Quarter' 'Apr' 'May' 'Jun'
Q3←'Third Quarter' 'Jul' 'Aug' 'Sep'
Q4←'Fourth Quarter' 'Oct' 'Nov' 'Dec'

CT←(c'1995'),Q1,Q2,Q3,Q4
CD←0,16p1 2 2 2

'F.G'WS('ColTitles'CT)('ColTitleDepth'CD)

```



1995											
First Quarter			Second Quarter			Third Quarter			Fourth Quarter		
Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
28	1	42	3	71	94	24	19	32	89	66	16
69	39	39	50	15	59	85	60	96	56	15	99
41	15	57	26	49	47	97	13	20	32	63	13
66	63	81	25	48	39	21	3	91	43	15	95
42	14	89	10	17	8	37	26	14	79	46	35
46	81	94	66	22	68	91	26	87	48	51	61

Note that the [Section 6.357](#) method is not supported in combination with hierarchical column titles.

6.117 ColTitleFCol Property

Applies To: [Section 6.263](#)

Description

The ColTitleFCol property specifies the colour of the column titles in a [Section 6.263](#) object

ColTitleFCol may be a scalar that specifies a single colour to be used for all of the column titles, or a vector that specifies the colour of each of the column titles

individually. An element of ColTitleFCol may be an enclosed 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or it may be a scalar that defines a standard Windows colour element (see [Section 6.45](#) for details). Its default value is 0 which obtains the colour defined for Button text.

6.118 ColTitles Property

Applies To: [Section 6.263](#), [Section 6.348](#)

Description

This property specifies the headings that are displayed above the columns in a [Section 6.263](#) object. If specified, it must be a vector of character vectors or matrices whose length is the same as the number of columns implied by the [Section 6.642](#) property. The default value of ColTitles is an empty character vector. In this case, the system displays "standard" spreadsheet column titles A-Z, AA-AZ, BA-BZ and so forth.

To disable the display of column titles in a [Section 6.263](#), you should set the [Section 6.618](#) property to 0.

6.119 ColumnClick Event 320

Applies To: [Section 6.348](#)

Description

If enabled, this event is reported when the user clicks on the column heading in a [Section 6.348](#) object. This event may not be disabled or affected by a callback function in any way.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ColumnClick' or 320
[3]	Column number	Integer
[4]	Button	button pressed (number) 1 = left button 2 = right button 4 = middle button
[5]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down

6.120 ColumnWidth Property

Applies To: [Section 6.346](#)

Description

This property specifies the column width in pixels of a multi-column [Section 6.346](#) object. See [Section 6.403](#) property for details.

6.121 Combo Object

Purpose: This object combines an input area with a list box and allows the user to enter a selection by typing text or by choosing an item from the list.

Parents ActiveXControl CoolBand Form Grid Group PropertyPage SubForm ToolBar ToolControl

Children Circle Cursor Ellipse Font Marker Poly Rect Text Timer

Properties Type Items Text Posn Size Style Coord Rows Border Active Visible Event VScroll HScroll SelItems SelText Sizeable Dragable FontObj FCol BCol CursorObj AutoConf Index Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex HasClearButton MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ChooseFont

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp

MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom
 GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap DropDown
 Select Change

Description

Three types of Combo box are provided by the [Section 6.577](#) property which may be 'Drop' (the default), 'Simple' or 'DropEdit'.

The [Section 6.334](#) property specifies the list of items which are displayed in the list box and from which the user can choose.

The [Section 6.519](#) property is a Boolean vector which specifies which (if any) of the items is selected. When the user chooses an item from the list, it is copied to the edit field and a [Section 6.513](#) event is generated. At this point you may use [Section 6.519](#) to identify the chosen item. You can also use [Section 6.519](#) to pre-select the contents of the edit field.

If the [Section 6.577](#) is 'Simple' or 'DropEdit', the user may type text into the edit field. In these cases, the contents of the edit field may also be specified or queried using the [Section 6.606](#) property.

Note that if the user first selects an item from the list box, then changes it in the edit field, the entry in the list box is automatically deselected. There is therefore no conflict between the value of [Section 6.605](#) and the value of [Section 6.519](#).

Warning

Windows truncates the contents of the edit field (reflected in the value of the Text property) to 510 characters.

For a Combo with [Section 6.577](#) 'Simple', the [Section 6.319](#) property specifies or reports the position of [Section 6.334](#) in the list box as a positive integer value. If [Section 6.319](#) has the value "n", it means that the "nth" item in [Section 6.334](#) is displayed on the top line in the list box. Note that [Section 6.319](#) can only be set using *Dyalog APL Language: Ws* and **not** by *Dyalog APL Language: Wc* and is ignored if all the [Section 6.334](#) fit in the list box. The default value for [Section 6.319](#) is $\square 10$.

The [Section 6.521](#) property identifies the portion of the edit field that is highlighted. It is not applicable to a Combo with [Section 6.577](#) 'Drop' as the user cannot enter or change data in its edit field.

The height of a Combo object with [Section 6.577](#) 'Drop' or 'DropEdit' is defined in a manner that is different from other objects. The height of the edit field is fixed, and is dependent only upon the size of the font. The height of the associated drop-down list box is determined by the [Section 6.494](#) property. The first element of the [Section 6.556](#) property (height) is ignored. For a Simple combo box (whose listbox is permanently displayed), the overall height is determined by the first element of [Section 6.556](#). [Section 6.494](#) is a "read-only" property.

The [Section 6.646](#) property specifies whether or not a vertical scrollbar is provided. The default value 0 means no scrollbar, setting [Section 6.646](#) to -1 or -2 specifies that the Combo has a vertical scrollbar.

If the [Section 6.577](#) is 'Simple' or 'DropEdit', the [Section 6.303](#) property determines whether or not the edit field may be scrolled. If [Section 6.303](#) is 0, the data is not scrollable, and the user cannot enter more characters once the field is full. If [Section 6.303](#) is -1 or -2 the field is scrollable, and there is no limit on the number of characters that can be entered. In neither case however is a horizontal scrollbar provided. If [Section 6.577](#) is 'Drop', the user is not allowed to enter data into the edit field anyway, and the value of [Section 6.303](#) is ignored.

[Section 6.646](#) and [Section 6.303](#) may only be set when the object is created and may not subsequently be changed.

Note that when you change the [Section 6.334](#) property using *Dyalog APL Language: Ws*, the [Section 6.605](#), [Section 6.519](#) and [Section 6.521](#) properties are all reset to their default values.

The Combo object will report a [Section 6.513](#) event (if enabled) when the user chooses an item from the list box. It will generate a [Section 6.85](#) event (if enabled) when the user manually alters the contents of the edit field and then changes the focus to another object.

6.122 ComboEx Object

Purpose: The ComboEx object is an extended version of the Combo object that provides additional features including item images

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm ToolBar ToolControl

Children Circle Cursor Ellipse Font Marker Poly Rect Text Timer

Properties Type Items Text Posn Size Style Coord Rows Border Active Visible Event Indents ImageListObj ImageIndex SelImageIndex CaseSensitive EditImage EditImageIndent PathWordBreak VScroll HScroll SelItems SelText Sizeable Dragable FontObj FCol BCol CursorObj AutoConf Index Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex HasClearButton MethodList ChildList EventList PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap DropDown Select

Description

The ComboEx object is a ComboBox that supports item images and indenting. It is a superset of the [Section 6.121](#) object and supports all its functionality. For further details, see [Section 6.121](#).

For most purposes, you can use the ComboEx object in place of the [Section 6.121](#) object whether or not you make use of the extended features of the ComboEx.

Like the basic [Section 6.121](#), the list of text items in the ComboEx is specified by the [Section 6.334](#) property. You may associate images with each of the text items using the [Section 6.317](#), [Section 6.315](#) and [Section 6.518](#) properties.

To do so, [Section 6.317](#) specifies the name of an ImageList object that contains a set of images. [Section 6.315](#) and [Section 6.518](#) map individual images from the ImageList to each of the text items specified by Items. [Section 6.315](#) specifies the image to be displayed when the item is not selected; [Section 6.518](#) specifies the image to be displayed when the item is selected.

The [Section 6.318](#) property specifies the amount by which each of the items are indented in units of 10 pixels

The appearance of the items is additionally controlled by the [Section 6.186](#) and [Section 6.187](#) properties. These are Boolean and their effect is summarised in the table below. Notice that Images are displayed only if both these properties are set to 1 (which is the default).

There are certain restrictions that apply to a ComboBox object with Style 'Simple', namely:

- images and indents do not apply to the edit control portion of the object.
- the object may not redraw properly if [Section 6.186](#) and/or [Section 6.187](#) are set to 0 or if [Section 6.67](#) or [Section 6.446](#) are set to 1.
- [Section 6.446](#) does not work.

	EditImageIndent	
EditImage	0	1
0	No images displayed, item text is indented as specified by Indents	No images displayed, item text is indented as specified by Indents plus the width of the images in ImageList
1	No images displayed, item text is indented as specified by Indents	Images are displayed, items are indented as specified by Indents

6.123 Configure Event 31

Applies To

Description

If enabled, this event is generated when the configuration of an object (position and/or size) is about to change.

For a [Section 6.227](#), the event is generated when the [Section 6.227](#) is resized or moved by the user.

For any object other than a [Section 6.227](#), it can occur in one of two ways. Firstly, whenever a [Section 6.227](#) is resized, the system (by default) re-arranges its children so as to maintain their relative position and size. This generates a Configure event (if enabled) for each one of them.

Secondly, it can occur as a result of the user resizing the object directly. This facility is enabled by setting the object's [Section 6.557](#) property to 1.

Note that a Configure event is **not** reported when an object is moved using "drag & drop". See [Section 6.173](#) (property) and [Section 6.174](#) (event) for details of this operation.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Configure' or 31
[3]	Y	y-position of top left corner
[4]	X	x-position of top left corner
[5]	H	height of object
[6]	W	width of object

For any object, the operation can be prevented by returning a scalar 0 from the callback function associated with the Configure event.

Full-Drag Considerations

The user may choose a system option, described here as *full-drag*, whereby the contents of the window are re-arranged during a resize operation.

If you manage the geometry of your controls using the Attach property, APL honours *full drag* during resize, changing the size and position of your controls dynamically for you.

However, if you manage the geometry of controls using Configure event callbacks, you should consider the following.

1. If full drag is in enabled, APL generates Configure events *during* the resize operation, allowing you to dynamically alter the geometry of controls as you wish. However, the following restrictions apply:
2. Configure callbacks will only be executed when the interpreter is *idle*. For example during a □□□ or during Session input (6-space prompt). If the user attempts to move/resize a window that has Configure callbacks attached when the interpreter is busy, the move/resize is not started. This is similar to the operation in non *full drag* mode, where the move/resize is allowed but the callback does not execute until the interpreter again becomes idle.
3. The callback cannot be traced. It is necessary to debug the callback code with *full drag* disabled.
4. Any untrapped errors in the Configure callback will not halt execution in the normal way, but will instead be reported in the *Status Window*. Note that it is also not possible to trap such errors higher up the SI stack than the Configure Callback.

5. There are some programming styles to be avoided if *full drag* Configure callbacks are to be processed correctly. For example events generated by monadic `□NQ` within a Configure callback will not be processed until the entire resize operation has been completed.
6. It is not possible to save a workspace from within a Configure Callback in *full drag* mode.

The above restrictions apply to Configure events when *full drag* is enabled, but *only* when *full drag* is enabled. The behaviour of Configure callbacks with *full drag* disabled is the same as for other events.

6.124 Container Property

Applies To: [Section 6.7](#)

Description

The Container property is a read-only property whose value is the `□OR` of an ActiveXContainer object that represents the *ActiveX Site* object of the application that is hosting the [Section 6.7](#).

The value of Container may be converted to a namespace using `□NS` or `□WC`.

The resulting object may then be used to obtain the values of ambient properties, or to access methods exposed by the host application via OLE interfaces. See [Section 6.418](#).

6.125 ContextMenu Event 410

Applies To

Description

If enabled, this event is reported when the user performs the standard Windows action to display a ContextMenu. These include clicking/releasing the right mouse button and pressing F10.

If the object has its own standard context menu, for example an Edit object, the default action is to display this menu. If the object is dockable (see [Section 6.162](#)), the default action is to display the standard (English) Dyalog APL docking menu.

You may use this event to display your own pop-up context menu, by `□DQ`'ing it within a callback function. In this case, your callback function should return 0 to disable the standard context menu.

The event message reported as the result of `WM_MOUSEMOVE`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ContextMenu' or 410
[3]	(reserved)	Empty character vector
[4]	Y	y-position of the mouse (number)
[5]	X	x-position of the mouse (number)

6.126 CoolBand Object

Purpose: The CoolBand object represents an area in a CoolBar that contains a child window.

Parents CoolBar

Children Bitmap BrowseBox Button Clipboard Combo ComboEx Cursor Edit FileBox Font Grid Group Icon ImageList Label List ListView Menu Metafile MsgBox OCXClass OLEClient Printer ProgressBar RichEdit Scroll Spinner Static StatusBar SubForm TabControl TCPSocket Timer TipField ToolBar ToolControl TrackBar TreeView UpDown

Properties Type Caption Posn Size Visible Event ImageIndex FCol BCol Picture Index Data KeepOnClose ChildEdge NewLine GripperMode Dockable UndocksToRoot MethodList ChildList EventList PropList

Methods Detach

Events Close Create DockStart DockMove DockRequest DockAccept DockEnd DockCancel

Description

The CoolBand object is a container object that represents a band in a [Section 6.127](#).

A CoolBand can have any combination of a gripper bar, a bitmap, a text label, and a single child object.

A CoolBand may not contain more than one child object, but that child object may itself be a container such as a [Section 6.624](#) or a [Section 6.578](#).

The [Section 6.66](#) property specifies a text string to be displayed to the left of the CoolBand. The colour of the text is specified by the [Section 6.205](#) property.

The [Section 6.315](#) property specifies an optional picture which is to be displayed alongside the [Section 6.66](#). If specified, [Section 6.315](#) is an index into an [Section 6.316](#) whose name is referenced via the [Section 6.317](#) property of the parent [Section 6.127](#).

The background in a CoolBand may be specified using its [Section 6.45](#) or [Section 6.447](#) properties. Although typically, the visible background area is small, it is visible through a [Section 6.628](#) [Section 6.624](#).

The [Section 6.91](#) property specifies whether or not the CoolBand leaves space above and below its child window.

The [Section 6.278](#) property specifies whether or not the CoolBand has a gripper bar which is used to reposition and resize the CoolBand within its parent [Section 6.127](#). [Section 6.278](#) may be 'Always' (the default), 'Never' or 'Auto'.

The position of a Cool Band within a [Section 6.127](#) is determined by its [Section 6.319](#) and [Section 6.409](#) properties, and by the position and size of preceding CoolBand objects in the same [Section 6.127](#). For a CoolBand, [Section 6.452](#) is a read-only property that reports its position but [Section 6.452](#) may not be used to set it.

The [Section 6.319](#) property specifies the position of a CoolBand within its parent [Section 6.127](#), relative to other CoolBands and is ☐IO dependent. Initially, the value of [Section 6.319](#) is determined by the order in which the CoolBands are created. You may re-order the CoolBands within a [Section 6.127](#), under program control, by changing [Section 6.319](#) with ☐ws.

The [Section 6.409](#) property specifies whether or not the CoolBand occupies the same row as an existing CoolBand, or is displayed on a new line within its [Section 6.127](#) parent. The value of [Section 6.409](#) in the first CoolBand in a [Section 6.127](#) is always 1, even if you specify it to be 0. You may move a CoolBand to the previous or next row by changing its [Section 6.409](#) property (using ☐ws) from 1 to 0, or from 0 to 1 respectively.

The 2nd element of the [Section 6.556](#) property determines the width of the CoolBand; the value of the 1st element is read-only.

[Section 6.556](#) may **only** be specified by ☐wc. However, when you create a CoolBand, it will automatically occupy all the available space in the current row, to the right of any preceding CoolBands. Only when you create *another* CoolBand in the *same* row, will the [Section 6.556](#) of the first CoolBand be honoured. The rightmost CoolBand will always extend to the right edge of the CoolBar, whatever its [Section 6.556](#).

If you create two or more CoolBands in the same row and you do not specify [Section 6.556](#), the first CoolBand will be maximised, and the others minimised.

When the user drags a CoolBand to a different row its [Section 6.319](#) and [Section 6.409](#) properties may change, as may the [Section 6.319](#) and [Section 6.409](#) properties of any another CoolBand that is affected by the operation.

If you wish to remember the user's chosen layout when your application terminates, you must store the values of [Section 6.319](#), [Section 6.556](#) and [Section 6.409](#) for each of the CoolBands. When your application is next started, you must re-create the CoolBands with the same values of these properties.

6.127 CoolBar Object

Purpose: The CoolBar object acts as a container for CoolBand objects.

Parents ActiveXControl Form

Children CoolBand ImageList Menu Timer

Properties Type Posn Size Align Event ImageListObj FCol BCol CursorObj Data Attach Handle KeepOnClose BandBorders DbClickToggle FixedOrder VariableHeight DockChildren Redraw MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help DockStart DockMove DockRequest DockAccept DockEnd DockCancel

Description

The CoolBar and [Section 6.126](#) objects provide an interface to Windows *Rebar Controls*

A CoolBar contains one or more bands ([Section 6.126](#) objects). Each band can have any combination of a gripper bar, a bitmap, a text label, and a single child object.

Using the gripper bars, the user may drag bands from one row to another, resize bands in the same row, and maximise or minimise bands in a row.

The following example illustrates a CoolBar containing 2 CoolBands each of which is displaying a ToolControl object.

```

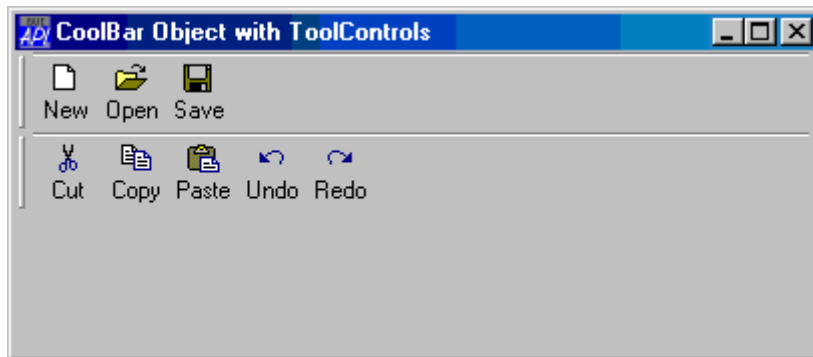
'F'WC'Form' 'CoolBar Object with ToolControls'('Size' 25 50)
'F.IL'WC'ImageList'('Masked' 0)('MapCols' 1)
'F.IL.'WC'Bitmap'('ComCtl32' 120)A STD_SMALL

'F.CB'WC'CoolBar'

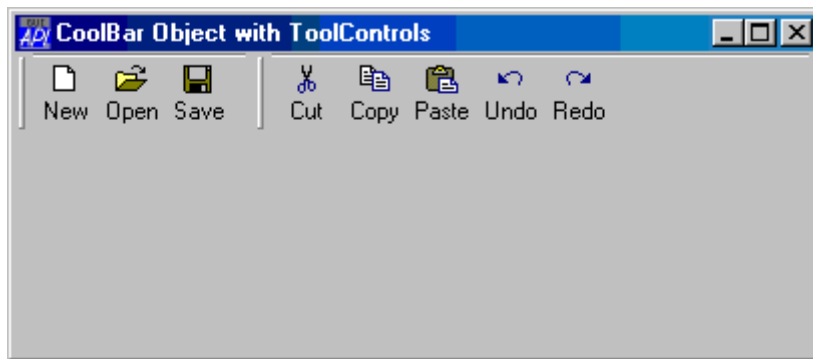
:With 'F.CB.C1'WC'CoolBand'
  'TB'WC'ToolControl'('ImageListObj' '#.F.IL')
  'TB.B1'WC'ToolButton' 'New'('ImageIndex' 7)
  'TB.B2'WC'ToolButton' 'Open'('ImageIndex' 8)
  'TB.B3'WC'ToolButton' 'Save'('ImageIndex' 9)
:EndWith

:With 'F.CB.C2'WC'CoolBand'
  'TB'WC'ToolControl'('ImageListObj' '#.F.IL')
  'TB.B1'WC'ToolButton' 'Cut'('ImageIndex' 1)
  'TB.B2'WC'ToolButton' 'Copy'('ImageIndex' 2)
  'TB.B3'WC'ToolButton' 'Paste'('ImageIndex' 3)
  'TB.B4'WC'ToolButton' 'Undo'('ImageIndex' 4)
  'TB.B5'WC'ToolButton' 'Redo'('ImageIndex' 5)
:EndWith

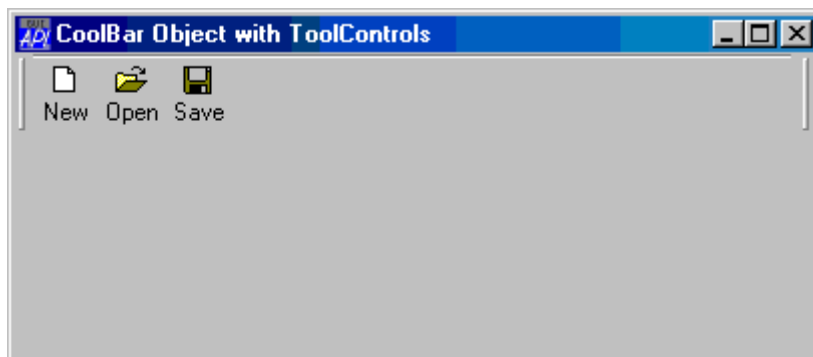
```



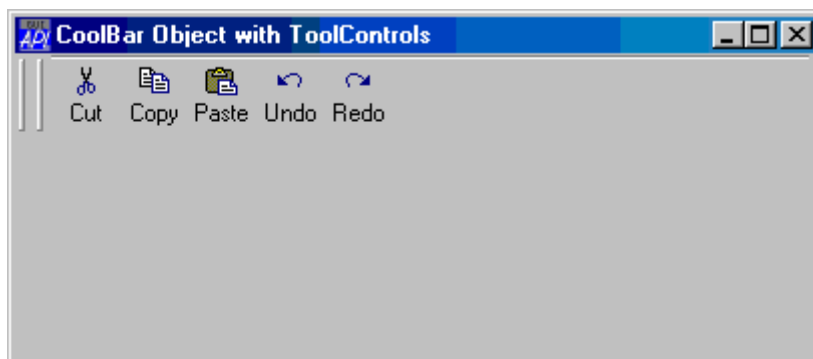
The CoolBar allows the user to organise the CoolBands within it as required. The next three pictures illustrate this feature.



after user has moved band 2 into row 1



after user has maximised band 1



after user has maximised band 2

The second example illustrates a CoolBar containing 3 CoolBands displaying an Edit, Combo and multi-line Edit respectively.

```

'F'WC'Form' 'CoolBar Object with simple controls'('Size' 25 40)
'F'WS'Coord' 'Pixel'

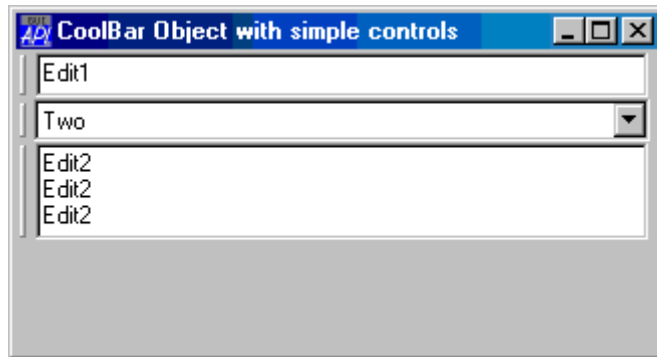
'F.CB'WC'CoolBar'

:With 'F.CB.C1'WC'CoolBand'
    'E1'WC'Edit' 'Edit1'
:EndWith

:With 'F.CB.C2'WC'CoolBand'
    'C1'WC'Combo'('One' 'Two' 'Three')('SelItems' 0 1 0)
:EndWith

:With 'F.CB.C3'WC'CoolBand'
    'E2'WC'Edit'(3 5p'Edit2')('Style' 'Multi')
:EndWith

```



The [Section 6.643](#) property specifies whether or not the CoolBar displays bands in different rows at the minimum required height (the default), or all the same height.

The [Section 6.43](#) property specifies whether or not narrow lines are drawn to separate adjacent bands. The default is 0 (no lines).

The [Section 6.145](#) property specifies whether or not the user must single-click (the default) or double-click to toggle a child [Section 6.126](#) between its maximised and minimised state.

The [Section 6.219](#) property specifies whether or not the CoolBar displays [Section 6.126](#) in the same order. If [Section 6.219](#) is 1, the user may move bands to different rows, but the band order is static. The default is 0. Note that when the user moves a CoolBand within a CoolBar, its Index and (potentially) [Section 6.409](#) properties will change to reflect its new position.

If you wish to display pictures in one or more of the [Section 6.126](#) owned by a CoolBar, you do so by setting the [Section 6.317](#) property to the name of an [Section 6.316](#) object which contains the pictures. Pictures are allocated to individual [Section 6.126](#) via their ImageIndex properties.

6.128 Coord Property

Applies To

Description

This property defines an object's co-ordinate system. It is a character string with one of the following values; 'Inherit', 'Prop', 'Pixel', 'RealPixel', 'ScaledPixel', 'User' or 'Cell' (graphics children of a [Section 6.263](#) only).

If Coord is 'Inherit', the co-ordinate system for the object is **inherited** from its parent. Note that the default value of Coord for the system object '.' is 'Prop', so by default all objects created by *Dyalog APL Language: Wc* inherit 'Prop'.

If Coord is 'Prop', the origin of the object's parent is deemed to be at its top left interior corner, and the scale along its x- and y-axes is 100. The object's position and size ([Section 6.452](#) and [Section 6.556](#) properties) are therefore specified and reported as a percentage of the dimensions of the parent object, or, for a [Section 6.227](#), of the screen.

If Coord is 'RealPixel', the origin of the object's parent is deemed to be at its top left interior corner, and the scale along its x- and y-axes is measured in physical pixel units. The object's position and size ([Section 6.452](#) and [Section 6.556](#) properties) are therefore reported and set in physical pixel units. If you set Coord on the system object to 'Pixel', the value of its [Section 6.556](#) property gives you the resolution of your screen. Note that pixels are numbered from 0 to (Size -1).

If Coord is 'ScaledPixel' the number of pixels specified for [Section 6.452](#), [Section 6.556](#), and other such properties will be automatically scaled by Dyalog APL according to the user's chosen display scaling factor. So if you specify an Edit object to be 80 pixels wide and 20 pixels high, and the user's scaling factor is 150%, Dyalog will automatically draw it 120 pixels wide and 30 pixels high. Dyalog will also de-scale coordinate values reported by `⌈WG` and event messages.

If Coord is 'Pixel', it is interpreted as either 'RealPixel' or 'ScaledPixel' according to the value of the **Dyalog_Pixel_Type** parameter, which is either ScaledPixel or RealPixel. See *Installation/Configuration: Dyalog Pixel Type*.

If this parameter is not specified, the default is `RealPixel`. So by default, when you set `Coord` to `Pixel`, it will be treated as `RealPixel`.

If `Coord` is `'User'`, the origin and scale of the co-ordinate system are defined by the values of the [Section 6.663](#) and [Section 6.661](#) properties of the **parent object**. Each of these is a 2-element numeric vector whose elements define the co-ordinates of top left and bottom right interior corners of the (parent) object respectively.

Note that if `Coord` is `'User'` and you change the values of [Section 6.663](#) and/or [Section 6.661](#) of the parent, the object (and all its siblings with `Coord 'User'`) are redrawn (and clipped) according to the new origin and scale defined for the parent. The values of their [Section 6.452](#), [Section 6.556](#) and [Section 6.449](#) properties are unaffected. Changing [Section 6.469](#) and/or [Section 6.469](#) therefore provides a convenient and efficient means to "**pan and zoom**".

The `Coord` property for graphic objects created as children of a `Grid` may also be set to `Cell`. Apart from being easier to compute, a graphic drawn using cell coordinates will expand and contract when the grid rows and columns are resized.

EXAMPLE

This statement creates a button 10 pixels high, 20 pixels wide, and 5 pixels down and along from the top-left corner of the parent [Section 6.227](#) T.

```
'T.B1' □WC 'Button' 'OK' (5 5) (10 20) ( 'Coord' 'Pixel' )
```

If you set `Coord` to `'RealPixel'` in the [Section 6.489](#) object `'.'`, then query its [Section 6.556](#), you get the dimensions of the screen in pixels, that is:

```
'.' □WS 'Coord' 'RealPixel'
'.' □WG 'Size'
480 640
```

If you set `Coord` to `'ScaledPixel'` in the [Section 6.489](#) object `'.'`, then query its [Section 6.556](#), you get the virtual resolution of the screen, that is:

```
'.' □WS 'Coord' 'ScaledPixel'
'.' □WG 'Size'
1080 1920
```

6.129 Copies Property

Applies To: [Section 6.455](#)

Description

Specifies the number of copies to be printed.

Copies is a non-zero scalar integer value whose default is defined by the current printer settings.

6.130 CornerTitleBCol Property

Applies To: [Section 6.263](#)

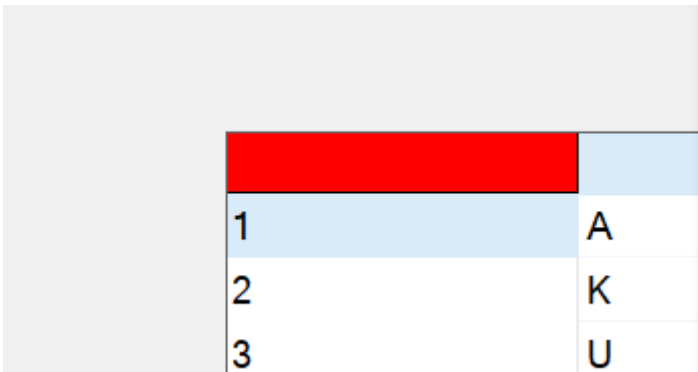
Description

This property specifies the colour used to fill the area in the left corner a [Section 6.263](#) . This is the rectangle above the row titles and to the left of the column titles.

CornerTitleBCol may be a 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or it may be a scalar that defines a standard Windows colour element (see [Section 6.45](#) for details). Its default value is 0 which means that the colour derives from your current Windows colour scheme.

EXAMPLE

```
f.g.CornerTitleBCol←255 0 0
```



6.131 Create Event 34

Applies To

Description

If enabled, this event is reported *after* an object has been created. You may not nullify or modify the event with a 0-returning callback, nor may you generate the event using `⌈NQ`, or call it as a method.

The event message reported as the result of `⌈DQ`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Create' or 34
[3]	Flag	1 = object was created by <code>⌈WC</code> 0 = object was created by)LOAD ,)COPY or <code>⌈OR</code>

This event also applies to the Session object `⌈SE` and may be used to fire a start-up function (in the `⌈SE` namespace) when APL initialises.

6.132 Cue Property

Applies To: [Section 6.56](#), [Section 6.185](#)

Description

This property specifies optional text to be displayed when a [Section 6.56](#) or an [Section 6.185](#) object is empty. For an [Section 6.185](#) object it applies only if the Style of the [Section 6.185](#) object is 'Single'.

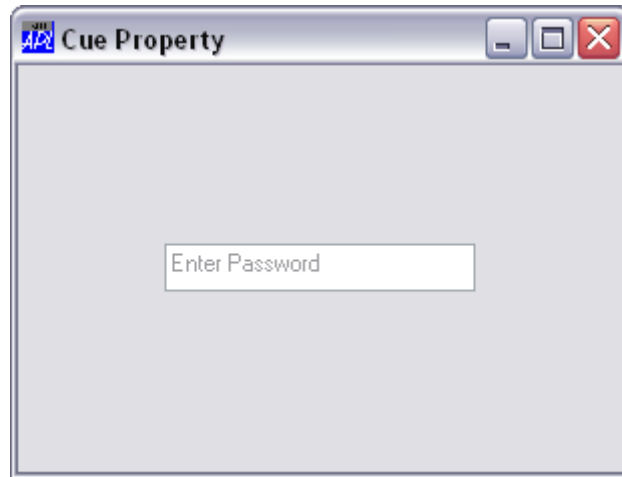
Note

This feature only applies if [Section 8.6](#) is enabled.

The Boolean property [Section 6.546](#) determines whether or not the cue should also be displayed once the user has tabbed into or clicked on the input field (and thus given it the focus).

EXAMPLE

```
'F' ⌈WC 'Form' 'Cue Property'
'F.E' ⌈WC 'Edit'
F.E.Cue←'Enter Password'
```



6.133 CurCell Property

Applies To: [Section 6.263](#)

Description

This property specifies or reports the current cell in a [Section 6.263](#) object. The current cell is the one that is currently addressed by the user. The current cell is identified visibly if it has an associated object, other than an external [Section 6.341](#) object.

6.134 CurrentColor Property

Applies To: [Section 6.108](#)

Description

The CurrentColor property is a 3-element integer vector that specifies and reports the RGB value of the currently selected colour in a [Section 6.108](#) object. Its default value is (0 0 0) which is black.

6.135 CurrentState Property

Applies To: [Section 6.604](#)

Description

The `CurrentState` property is a read-only property that reports the current state of a [Section 6.604](#) object. Its possible values and their means are as follows:

CurrentState	Description
'Open'	a client socket that is not yet connected or a UDP socket
'Bound'	a server socket that has been bound
'Listening'	a server socket to which a client has not yet connected
'Connected'	a client or server socket that is connected
'IHaveClosed'	a temporary state on the way to Closed
'PartnerHasClosed'	a temporary state on the way to Closed
'Closed'	a socket that has been closed by both client and server

6.136 Cursor Object

Purpose: This object defines a cursor.

Parents ActiveXControl Animation Button Calendar Combo ComboEx CoolBand DateTimePicker Edit Form Grid Group ImageList Label List ListView OLEServer ProgressBar PropertyPage PropertySheet RichEdit Root Scroll SM Static StatusBar SubForm TCPSocket ToolBar ToolControl TrackBar TreeView UpDown

Children Timer

Properties Type File Bits CMap Mask HotSpot KeepBits Event Data Handle Accelerator KeepOnClose MethodList ChildList EventList PropList

Methods Detach FileRead FileWrite

Events Close Create Select

Description

The [Section 6.207](#) property defines the name of a cursor file associated with the Cursor object, or it specifies the name of a DLL and the resource number or name of the cursor therein. If you omit the file extension, the system assumes .CUR. To use an animated cursor you must therefore specify the .ANI extension explicitly.

If the value of the [Section 6.207](#) property is set by *Dyalog APL Language: Ws*, no immediate action is taken, but the corresponding file may subsequently be read or written using the [Section 6.212](#) or [Section 6.213](#) methods.

The [Section 6.48](#) and [Section 6.366](#) properties define the appearance of the cursor. Both are Boolean matrices with a shape of 32 32. The colour of each pixel in the cursor is defined by the following table. Note that a 0 in [Section 6.48](#) combined with a 1 in [Section 6.366](#) causes the corresponding pixel to be the colour of the background. This is used to give the cursor a non-rectangular shape.

Bits	0	1	0	1
Mask	0	0	1	1
Pixel	Black	White	Background	Inverse

The [Section 6.301](#) property determines the point within the cursor that registers its position over another object.

A Cursor is **used** by setting the [Section 6.137](#) property of another object to its name or ref.

6.137 CursorObj Property

Applies To

Description

This property is used to associate a particular cursor with an object.

Its value is either a simple scalar number which specifies a standard Windows cursor, or the name of, or ref to, a [Section 6.136](#) object. The standard Windows cursors are :

0	arrow (Windows default)
1	hourglass
2	crosshair
3	I-Beam
4	crossing vertical/horizontal double-headed arrows
5	diagonal double-headed arrows (left-to-right)
6	vertical double-headed arrows
7	diagonal double-headed arrows (right-to-left)
8	horizontal double-headed arrows
9	upward pointing arrow
10	box
11	crossing vertical/horizontal double-headed arrows
12	no-entry sign
13	arrow with hourglass
14	pointing hand

If CursorObj is set to anything other than an empty vector (which is the default) it defines the appearance of the cursor when the mouse pointer is moved into the object. If CursorObj is an empty vector, the shape of the cursor remains unchanged when the mouse pointer enters the object. In effect, the cursor is "inherited" from its parent. Exceptions to this rule are certain objects which have special cursors by default.

If the value of CursorObj for the [Section 6.482](#) object is set to anything other than an empty vector, it applies to **all** [Section 6.227s](#) and their children, irrespective of their own CursorObj values. Therefore, if you want to indicate that your application is "working" and is not responsive to input, you can simply do:

```
',' WS 'CursorObj' 1 A Hourglass cursor
```

Then to reset the application you do:

```
',' WS 'CursorObj' ''
```


6.138 CustomColors Property

Applies To: [Section 6.108](#)

Description

The CustomColors property is a 1-row, 16-column nested matrix which specifies the RGB values of the colours displayed in the *Custom colors* section of the Windows colour selection dialog box when displayed by a [Section 6.108](#) object.

Each element of CustomColors is a 3-element integer vector specifying an RGB colour value.

By default, each element of CustomColors is (0 0 0). If the user selects a new custom colour from the Windows colour selection dialog box, its value will be reported by CustomColors. CustomColors must always have shape (1 16).

Note that CustomColors is maintained separately for each separate [Section 6.108](#), and CustomColors defaults to (1 16 p=0 0 0) for each new [Section 6.108](#) that you create. If you want to maintain a global custom colour table for your application, you must do this yourself.

6.139 CustomFormat Property

Applies To: [Section 6.143](#)

Description

Specifies a custom format for the date/time display in a [Section 6.143](#).

CustomFormat is a character vector that may contain a mixture of date/time format elements and body text. The date/time elements are replaced by the actual date/time values when the object is displayed. The body text is displayed *as-is*. Note that CustomFormat may only be specified when the [Section 6.143](#) object is created.

The date/time elements are defined by the following groups of characters, notice that they are case-sensitive:

Element	Description
d	The one- or two-digit day.
dd	The two-digit day. Single-digit day values are preceded by a zero.
ddd	The three-character weekday abbreviation.
dddd	The full weekday name.
h	The one- or two-digit hour in 12-hour format.
hh	The two-digit hour in 12-hour format. Single-digit values are preceded by a zero.
H	The one- or two-digit hour in 24-hour format.
HH	The two-digit hour in 24-hour format. Single-digit values are preceded by a zero.
m	The one- or two-digit minute.
mm	The two-digit minute. Single-digit values are preceded by a zero.
M	The one- or two-digit month number.
MM	The two-digit month number. Single-digit values are preceded by a zero.
MMM	The three-character month abbreviation.
MMMM	The full month name.
t	The one-letter AM/PM abbreviation (that is, AM is displayed as "A").
tt	The two-letter AM/PM abbreviation (that is, AM is displayed as "AM").
yy	The last two digits of the year (that is, 1996 would be displayed as "96").
yyyy	The full year (that is, 1996 would be displayed as "1996").

The body text is defined by sub-strings contained within single quotes. For example, to display the current date with the format "Today is: 04:22:31 Tuesday Mar 23, 1996", the format string is defined as follows:

```
CustomFormat
'Today is: 'hh':'m':'s dddd MMM dd', 'yyyy'
```

To include a single quote in your body text, use two consecutive single quotes. For example, to produce output that looks like: "Don't forget Mar 23, 1996", CustomFormat should be specified as follows:

```
CustomFormat
'Don''t forget' MMM dd', ' yyyy
```

Note

Non-format characters that are not delimited by single quotes will result in unpredictable display by the [Section 6.143](#) object.

6.140 Data Property

Applies To

Description

This property allows you to associate arbitrary data with an object. The value of the Data property may be any APL array.

6.141 DateTime Property

Applies To: [Section 6.143](#)

Description

Specifies the value of date/time in a [Section 6.143](#).

The DateTime property represents the date and time value that is currently displayed in a [Section 6.143](#) object.

It is normally a 4-element integer vector containing the date (as an IDN), hour, minutes and seconds respectively.

However, if the checkbox shown in the object is unset (see [Section 6.284](#)), the value of DateTime will be 0 (zilde).

6.142 DateTimeChange Event 267

Applies To: [Section 6.143](#)

Description

If enabled, this event is reported by a [Section 6.143](#) object when the user changes the [Section 6.141](#) value. This occurs when the user selects a new date from the drop-down calendar, or increments or decrements a date time element using the spinner buttons, or edits a datetime element using the keyboard. In the latter case, the event may not be generated until the input focus leaves the corresponding date time element.

The event message reported as the result of `DDQ`, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DateTimeChange' or 267
[3]	IDN	integer
[4]	Hour	integer
[5]	Minute	integer
[6]	Second	integer

This event is reported for information only and cannot be disabled or modified in any way. It is not possible to enqueue this event; set the DateTime value instead.

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

6.143 DateTimePicker Object

Purpose: The DateTimePicker object is an editable date/time field with an optional drop-down Calendar.

Parents ActiveXControl Form Grid Group PropertyPage SubForm ToolBar

Children Cursor Font Menu MsgBox TCPsocket Timer

Properties Type Posn Size Style Coord Align Border Active Visible Event DateTime MinDate MaxDate CalendarCols Today HasToday CircleToday WeekNumbers MonthDelta HasCheckBox FieldType CustomFormat Sizeable Draggable FontObj

CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Translate
 Accelerator AcceptFiles KeepOnClose Redraw TabIndex MethodList ChildList EventList
 PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj
 IDNToDate DateToIDN

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles
 DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp
 MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom
 GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap
 DateTimeChange DropDown CloseUp Select

Description

The DateTimePicker object represents the built-in Windows date and time picker control. For most purposes, the DateTimePicker supersedes the use of Label, Edit and Spinner objects for displaying and entering dates and times. Unlike the Edit and Spinner objects, it is not possible for the user to enter an invalid date or time into a DateTimePicker.

The Style property may be either 'Combo' (the default) or 'UpDown'. The former provides a drop-down calendar that behaves in the same way as the Calendar object and whose appearance and behaviour is controlled by a set of properties namely CalendarCols, CircleToday, HasToday, MaxDate, MinDate, MonthDelta, Today and WeekNumbers that are common to the Calendar. Note that the Style property may only be set when the object is created.

If Style is 'Combo', the Align property specifies the horizontal alignment of the drop-down Calendar which may be 'Left' (the default) or 'Right'.

If Style is 'UpDown', the DateTimePicker includes instead a pair of spinner buttons that allow the user to increment and decrement values in the various sub-fields provided by the control.

The [Section 6.141](#) property represents the date and time value that is currently displayed in the object. This is a 4-element vector containing the IDN, hour, minutes and seconds respectively.

The FieldType property specifies one of a set of pre-defined date/time formats to be used by the control. This is a character vector that may be empty (the default), 'Date', 'DateCentury', 'LongDate', 'Time' or 'Custom'. Specifying an empty vector is the same as specifying 'Date'. Note that 'DateCentury' always displays a 4-digit year, regardless of the user's Windows settings.

If `FieldType` is set to `'Custom'`, the format is defined by the [Section 6.139](#) property. `CustomFormat` is a character vector that may contain a mixture of date/time format elements and body text.

The [Section 6.284](#) property is a Boolean value (default 0) that specifies whether or not a checkbox is displayed in the object. This allows the user to specify whether or not the date/time displayed in the `DateTimePicker` is applicable.

6.144 DateToIDN Method 264

Applies To: [Section 6.58](#), [Section 6.143](#), [Section 6.489](#)

Description

This method is used to convert a date from `□TS` format into an [Section 8.5](#) suitable for use in a [Section 6.58](#) object.

The argument to `DateToIDN` is a 3-element array as follows:

[1]	Year	Integer
[2]	Month	Integer
[3]	Day	Integer

`DateToIDN` will also accept a single enclosed argument containing these values. In either case, if you specify more than 3 numbers, excess elements they will be ignored.

EXAMPLES

```
F.C.DateToIDN 1998 9 11
36048
F.C.DateToIDN <1998 9 11
36048
F.C.DateToIDN □TS
36048
F.C.DateToIDN,□TS
36048
```

6.145 DbClickToggle Property

Applies To: [Section 6.127](#)

Description

The DbClickToggle property specifies whether or not the user must single-click or double-click to toggle a child [Section 6.126](#) between its maximised or minimised state.

DbClickToggle is a single number with the value 0 (single-click toggles state) or 1 (double-click toggles state); the default is 0.

6.146 DDE Event 50

Applies To: [Section 6.489](#)

Description

If enabled, a DDE event is generated whenever a DDE message is received by Dyalog APL. This will occur whenever a server notifies APL that the value of a shared variable has changed, and whenever a client application requests data from APL. If you have several shared variables, you can determine which of them has changed or whose value has been requested using `⊞SVs`.

This event **only** applies to the [Section 6.489](#) object ".", so to enable it you must execute one of the following statements :

```

'.' ⊞WS 'Event' 50 1
'.' ⊞WS 'Event' 50 fn
'.' ⊞WS 'Event' 50 fn larg

```

The first statement would cause *Dyalog APL Language: Dq* to terminate on receipt of a DDE event. The second would cause it to call "fn" each time. The third would do likewise but the value in "larg" would be supplied as its left argument.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DDE ' or 50

Note

Due to the nature of DDE "conversations" messages may be received when in fact no change in the value of any shared variables has occurred. Your application code must therefore be prepared to cater for this situation.

6.147 Decimals Property

Applies To: [Section 6.56](#), [Section 6.185](#), [Section 6.341](#), [Section 6.563](#)

Description

This property specifies the number of decimal places to which a number is to be displayed in an [Section 6.185](#) or [Section 6.341](#) object with [Section 6.206](#) 'Numeric'. For an [Section 6.185](#) object, Decimals also specifies the maximum number of digits that the user may enter after a decimal point.

The default value of decimals is -1 which allows any number of decimal places to be entered.

6.148 Default Property

Applies To: [Section 6.55](#), [Section 6.399](#)

Description

This property determines which of a set of push buttons in a [Section 6.227](#), [Section 6.578](#) or [Section 6.399](#) is the default button.

In a [Section 6.227](#) or [Section 6.578](#), the Default [Section 6.55](#) will generate a [Section 6.513](#) event (30) when the user presses the Enter key, even though the Default [Section 6.55](#) may not have the focus at the time.

If however, the user explicitly shifts the focus to another Push [Section 6.55](#), the automatic selection of the Default [Section 6.55](#) is disabled and the Enter key applies to the [Section 6.55](#) with the focus.

For a [Section 6.55](#), the Default property has the value 1 or 0. As only one [Section 6.55](#) can be the Default [Section 6.55](#), setting Default to 1 for a particular [Section 6.55](#) automatically sets Default to 0 for all others with the same parent.

In a [Section 6.399](#), Default specifies which button initially has the focus. It has the value 1, 2 or 3 corresponding to the three buttons that can be defined. See [Section 6.54](#) property for further details.

6.149 DefaultColors Property

Applies To: [Section 6.108](#)

Description

The DefaultColors property is a nested matrix which specifies the RGB values of the colours shown in the colour selection drop-down displayed by a [Section 6.108](#) object.

The shape of DefaultColors determines the number of rows and columns in the colour selection drop-down.

Each element of DefaultColors is a 3-element integer vector specifying an RGB colour value.

6.150 DelCol Method 155

Applies To: [Section 6.263](#)

Description

This method is used to delete a specified column from a [Section 6.263](#) object.

The argument to DelCol is a 1 or 2-element vector as follows:

[1]	Column number	number of the column (integer) to delete
[2]	Undo flag	0 or 1 (optional; default 0)

If the *Undo flag* 1, the column may subsequently be restored by invoking the [Section 6.635](#) method. If the *Undo flag* is omitted or is 0, the operation may not be undone.

6.151 DelComment Method 221

Applies To: [Section 6.263](#)

Description

This method is used to delete a comment from a [Section 6.263](#).

The argument to DelComment is a 2 array as follows or $\emptyset$:

[1]	Row	integer
[2]	Column	integer

For example, the following expression removes the comment associated with the cell at row 2, column 1.

```
F.C.DelComment 2 1
```

Note that to delete a comment associated with a row or column *title*, the appropriate element in the argument should be -1 .

If the argument is $\emptyset$, all comments are deleted.

6.152 DeleteChildren Method 311

Applies To: [Section 6.629](#)

Description

This method is used to delete child items from a parent item in a [Section 6.629](#) object.

The argument to DeleteChildren is a scalar or 1 element array as follows:

[1]	Item number	Integer.

Item number specifies the index of the parent item from which the child items are to be removed.

The result is an integer that indicates the number of children that have been removed from the parent item.

6.153 DeleteItems Method 309

Applies To: [Section 6.629](#)

Description

This method is used to delete items from a [Section 6.629](#) object.

The argument to DeleteItems is a 2-element array as follows:

[1]	Item number	Integer.
[2]	Number of Items	Integer.

Item number specifies the index of the first item to be removed.

Number of items specifies the number of items to be removed and refers to those items *at the same level* in the [Section 6.629](#) hierarchy as the *Item number*. *Number of items* is optional and defaults to 1.

Note that any children of these items will also be removed.

The result is an integer that indicates the total number of items, including children, that have been removed from the [Section 6.629](#).

6.154 DeleteTypeLib Method 521

Applies To: [Section 6.489](#)

Description

The DeleteTypeLib method removes a loaded Type Library from the workspace.

The argument to DeleteTypeLib is as follows:

[1]	TypeLib	character vector

The Type Library may be identified by its name or by its class id.

The result is 0, 1 or -1.

If successful, the specified Type Library, and all dependent Type Libraries not referenced by any other currently loaded Type Libraries, are removed from the active workspace. The result is 1.

If the specified Type Library is in use, no action is taken and the result is 0.

If the argument is not the name or CLSID of a loaded Type Library, no action is taken and the result is -1.

6.155 DelRow Method 154

Applies To: [Section 6.263](#)

Description

This method is used to delete a specified row from a [Section 6.263](#) object.

The argument to DelRow is a 1 or 2-element array as follows:

[1]	Row number	number of the row (integer) to delete
[2]	Undo flag	0 or 1 (optional; default 0)

If the *Undo flag* is 1, the column may subsequently be restored by invoking the [Section 6.635](#) method. If the *Undo flag* is omitted or is 0, the operation may not be undone.

6.156 Depth Property

Applies To: [Section 6.629](#)

Description

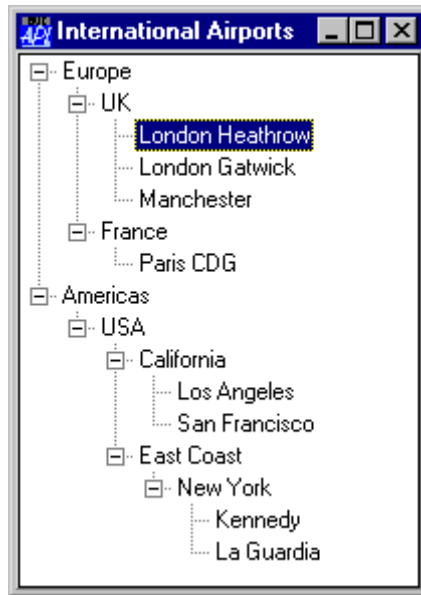
The Depth property specifies the structure of the items in a [Section 6.629](#) object. It is either a scalar 0 or an integer vector of the same length as the Items property.

A value of 0 indicates that the corresponding item is a top-level item. A value of 1 indicates that the corresponding item is a child of the most recent item whose Depth is 0; a value of 2 indicates that the corresponding item is a child of the most recent item whose Depth is 1, and so forth.

For example:

AIRPORTS	DEPTH	Description
Europe	0	Top-level (root) item
UK	1	1st sub-item of Europe
London Heathrow	2	1st sub-item of UK
London Gatwick	2	2nd sub-item of UK
Manchester	2	3rd sub-item of UK
France	1	2nd sub-item of Europe
Paris CDG	2	1st sub-item of France
Americas	0	Top-level (root) item
USA	1	1st sub-item of N.America
California	2	1st sub-item of USA
Los Angeles	3	1st sub-item of California
San Francisco	3	2nd sub-item of California
East Coast	2	2nd sub-item of USA
New York	3	1st sub-item of East Coast
Kennedy	4	1st sub-item of NY
La Guardia	4	2nd sub-item of NY

```
'F'WC'FORM' 'International Airports'  
'F.TV'WC'TreeView'AIRPORTS(0 0)(100 100)('Depth'DEPTH)
```



6.157 Detach Method 270

Applies To

Description

This method is used to detach the GUI component from an object without losing the functions, variables and sub-namespaces that it may contain.

The Detach method is niladic.

The effect of this method is to remove the GUI component associated with the named object, leaving behind a plain namespace of the same name. All non-GUI child objects are retained. GUI child objects are either destroyed, or similarly converted to plain namespaces depending upon the values of their KeepOnClose properties.

6.158 DevCaps Property

Applies To: [Section 6.455](#), [Section 6.489](#)

Description

This property reports the device capabilities of the screen or printer. It is a 4-element nested vector as follows:

[1]	Height and Width:2-element numeric vector of device in pixels
[2]	Height and Width:2-element numeric vector of device in mm
[3]	Number of colours or ~ 1
[4]	Windows scaling factor as a percentage (100=no scaling). This value is the same as reported in the Display section of the Windows Control Panel

This property is useful if you want to make objects of a specific physical size. For example, to draw a 10mm square in a [Section 6.227](#) 'F' at (5,5):

```
Size ← 10× ÷÷/2↑'. ' □WG 'DevCaps '
'F.R' □WC 'Rect' (5 5) Size ('Coord' 'Pixel')
```

Notes

- the physical size reported for the screen is typically only a *nominal* size, because, if you use a generic video driver, Windows has no way to tell what size of screen is attached to your computer.
- The number of colours is reported only if the device has a colour depth of no more than 8 bits per pixel. For devices with greater colour depths, ~ 1 is returned.
- new elements may be added to DevCaps in future releases.

6.159 Directory Property

Applies To: [Section 6.208](#)

Description

The Directory property contains a simple character vector which specifies the initial directory from which a list of suitable files is displayed.

If, whilst interacting with the [Section 6.208](#), the user changes directory and exits by pressing "OK" or by closing the [Section 6.208](#), the value of the Directory property is updated accordingly.

6.160 DisplayChange Event 137

Applies To: [Section 6.489](#)

Description

If enabled, this event is reported when the user changes the screen resolution or number of colours. The event is reported for information only; you cannot prevent the change from occurring.

The event message reported as the result of `DisplayChange`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DisplayChange' or 137
[3]	Height	Integer. Number of pixels in the y-direction
[4]	Width	Integer. Number of pixels in the x-direction
[5]	Number of colours	Integer.

6.161 Divider Property

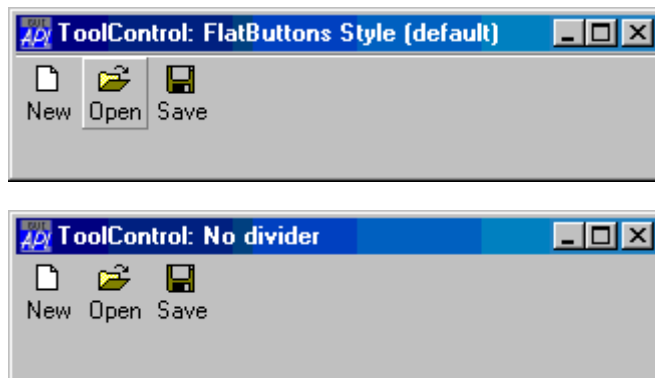
Applies To: [Section 6.624](#)

Description

The Divider property controls the presence or absence of a recessed line drawn above, below, to the left of, or to the right of a [Section 6.624](#) object.

Divider is a single number with the value 0 (dividing line is *not* drawn) or 1 (a dividing line *is* drawn); the default is 1.

The pictures below illustrates a [Section 6.624](#) drawn with and without a divider.



6.162 Dockable Property

Applies To: [Section 6.126](#), [Section 6.227](#), [Section 6.578](#), [Section 6.624](#)

Description

The Dockable property specifies whether or not an object may be docked or undocked.

Dockable is a character vector containing 'Never' (the default), 'Always' or 'Disabled'.

If Dockable is 'Never', the object may not be docked or undocked by the user, and the docking menu items are not present in the object's context menu. This is the default.

If Dockable is 'Always', the object may be docked or undocked by the user, and the docking menu items are present in the object's context menu.

If Dockable is 'Disabled', the object may not currently be docked or undocked by the user, but the docking menu items are present in the object's context menu.

Note that by default, the user may switch between Dockable 'Always' and 'Disabled' by toggling the *Dockable* menu item. If you want to exercise full control over this property, you may implement your own context menu (see [Section 6.125](#))

6.163 DockAccept Event 483

Applies To: [Section 6.126](#), [Section 6.127](#), [Section 6.227](#), [Section 6.578](#), [Section 6.624](#)

Description

If enabled, this event is reported by a host object just before it accepts a client object docking operation. This event is reported (by the host) immediately after the [Section 6.169](#) is reported (by the client).

The event message reported as the result of `DDQ`, or supplied as the right argument to your callback function, is a 7-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DockAccept ' or 483
[3]	Client Object	ref or character vector
[4]	Edge	character vector
[5]	y-position	number
[6]	x-position	number
[7]	Outline rectangle	4-element nested

Elements 4-7 of this event message are the same as those reported by [Section 6.168](#), and the effect of a callback function is identical. See [Section 6.168](#) for further information.

6.164 DockCancel Event 485

Applies To: [Section 6.126](#), [Section 6.127](#), [Section 6.227](#), [Section 6.578](#), [Section 6.624](#)

Description

If enabled, this event is reported by a client object when the user aborts a docking operation by pressing Escape.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DockCancel ' or 485

This event is reported for information only and cannot be cancelled or inhibited in any way.

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

6.165 DockChildren Property

Applies To: [Section 6.127](#), [Section 6.227](#), [Section 6.578](#)

Description

The DockChildren property specifies the names of client objects that may be docked in a host object.

DockChildren may be a single ref or simple character scalar or vector, or a vector of refs or character vectors. Each item represents an object that may be docked. Notice that if you use a name, you must specify the simple name of the object, excluding any part of its full pathname that refers to a parent; that is, the specified names must not contain any leading pathname information.

If the name of, or ref to, a [Section 6.162](#) object occurs in the DockChildren property, the host object will generate DockMove events when the client is dragged over it, and will generate a DockAccept event when a docking operation takes place.

If the name of, or ref to, the client object is not present in its DockChildren property, the object will not respond in any way as the client is dragged over it.

The following example shows the creation of 3 dockable forms, all of which are dockable in a host form called h1.

The first, c1, is a totally independent Form. When docked in h1, it will become a SubForm h1.c1. When undocked, it will revert to an independent Form c1.

The second, c2, is created initially as a child of h1 and will therefore be displayed above it in the window stacking order. When docked it will become a SubForm h1.c2. When undocked, it will revert back to a dependent Form h1.c2. In all cases, it appears on top of h1.

The third, c3, is created initially as a child of another Form, h2. When docked (in h1) it will become a SubForm h1.c3. When undocked, it will become a dependent Form h1.c3, and will therefore appear above h1 in the stacking order.

```
'h1' □WC 'Form' 'Host1'
'h2' □WC 'Form' 'Host2'

'c1' □WC 'Form' 'Client 1' ('Dockable' 'Always')
'h1.c2' □WC 'Form' 'Client 2' ('Dockable' 'Always')
'h2.c3' □WC 'Form' 'Client 3' ('Dockable' 'Always')

h1.DockChildren←'c1' 'c2' 'c3'
```

6.166 Docked Property

Applies To: [Section 6.227](#), [Section 6.578](#)

Description

The Docked property is a read-only property that indicates whether or not an object is currently docked.

Docked is a single number with the value 0 (is not docked) or 1 (is docked).

6.167 DockEnd Event 484

Applies To: [Section 6.126](#), [Section 6.127](#), [Section 6.227](#), [Section 6.578](#), [Section 6.624](#)

Description

If enabled, this event is reported by a client object after it has been successfully docked in a host object.

The event message reported as the result of `□□DQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DockEnd' or 484

This event is reported for information only and cannot be cancelled or inhibited in any way.

6.168 DockMove Event 481

Applies To: [Section 6.126](#), [Section 6.127](#), [Section 6.227](#), [Section 6.578](#), [Section 6.624](#)

Description

If enabled, this event is reported by a host object when a dockable object (the client) is dragged over it. The event will only be reported if the name of the client object is included in the list of objects that the host object will accept, which is defined by its DockChildren property.

The event message reported as the result of `□□DQ`, or supplied as the right argument to your callback function, is a 7-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DockMove' or 481
[3]	Client Object	ref or character vector
[4]	Edge	character vector
[5]	y-position	number
[6]	x-position	number
[7]	Outline rectangle	(see below)

The 4th element of the event message Edge is a character vector that indicates along which edge of the host object the client object will be docked if the mouse button is released. It is either 'Top', 'Bottom', 'Left', 'Right' or 'None'. The latter indicates that the object will not be docked. An object will dock only if the mouse pointer is inside, and sufficiently near to an edge of, the host.

The 5th and 6th elements of the event message report the position of the mouse pointer in the host object.

The 7th element of the event message is a 4-element nested vector containing the y-position, x-position, height and width of a rectangle. If Edge is 'None', this is the bounding rectangle of the client object. Otherwise, the rectangle describes a docking zone in the host that the client object will occupy when the mouse button is released.

If a callback function returns 0, the system displays the bounding rectangle and not a docking zone, and the docking operation is inhibited. You could use this mechanism to prohibit docking along one or more edges, whilst allowing it along others.

A callback function may modify the event message to cause a different sized docking zone to be displayed, or to force docking along a particular edge.

The DockMove event is generated repeatedly as the docking object is dragged.

6.169 DockRequest Event 482

Applies To: [Section 6.126](#), [Section 6.127](#), [Section 6.227](#), [Section 6.578](#), [Section 6.624](#)

Description

If enabled, this event is reported by a client object just before it is docked in a host object, when the user releases the mouse button.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 7-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DockRequest' or 482
[3]	Host Object	ref or character vector
[4]	Edge	character vector
[5]	y-position	number
[6]	x-position	number
[7]	Outline rectangle	4-element nested

Elements 4-7 of this event message are the same as those reported by [Section 6.168](#), and the effect of a callback function is identical. See [Section 6.168](#) for further information.

6.170 DockShowCaption Property

Applies To: [Section 6.227](#), [Section 6.578](#)

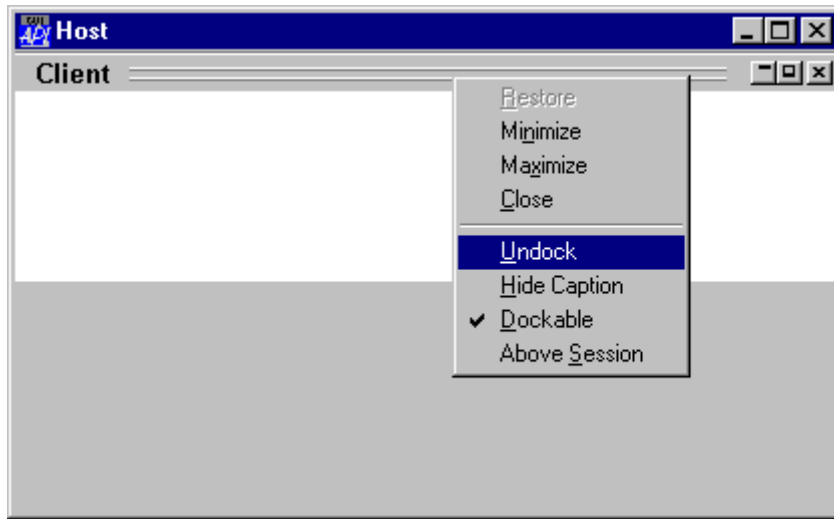
Description

The DockShowCaption property specifies whether or not a Form displays a title bar when it is docked as a SubForm.

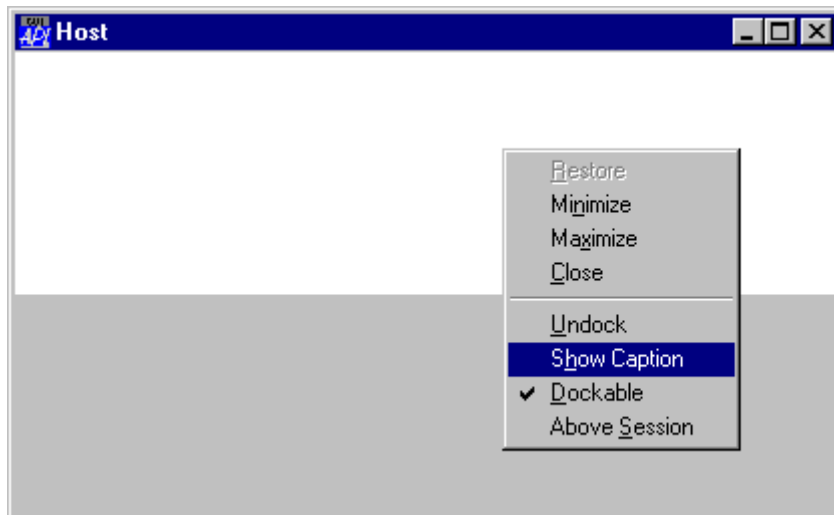
DockShowCaption is a single number with the value 0 or 1 (the default).

The DockShowCaption property may be toggled on and off by the user from the object's context menu.

The first picture below illustrates a Form, docked as a SubForm, whose DockShowCaption property is 1, but is about to be set to 0.



The next picture shows the same docked Form with DockShowCaption set to 0.



6.171 DockStart Event 480

Applies To: [Section 6.126](#), [Section 6.127](#), [Section 6.227](#), [Section 6.578](#), [Section 6.624](#)

Description

If enabled, this event is reported by a dockable object (one whose [Section 6.162](#) property is set to 1) when the user starts to drag it using the mouse.

The event message reported as the result of `□DQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DockStart' or 480

A callback function may prevent the docking operation from starting by returning 0.

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

6.172 DoPopup Event 846

Applies To: [Section 6.307](#)

Description

This event is triggered when the [Section 6.307](#) client attempts to open a new window. This could be fired by an HTML `<a>` tag with the target attribute set to open a URL in a new window or by a JavaScript `window.open()` call. Note that this does not apply to JavaScript Popup Boxes.

By default the HTMLRenderer ignores a request for a new window, but if the DoPoup event, is enabled, it provides the information needed to process the request in the workspace.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'DoPopup' or 846
[3]	URL	the requested url
[4]	Attributes	requested window attributes (see below)
[5]	Framename	character vector framename

Attributes is a 7-element nested vector that specifies the requested attributes for the new window. The HTMLRenderer currently provides no mechanism to use this information.

[1]	2-element vector of top, left positions – positions not specified are 0
[2]	2-element vector of height, width – sizes not specified are 0
[3]	Integer "WindowDisposition". See https://magpcss.org/ceforum/apidocs3/projects/(default)/cef_window_open_disposition_t.html .
[4]	Boolean menubar (default=1)
[5]	Boolean scrollbar (default=1)
[6]	Boolean statusbar (default=1)
[7]	Boolean location/toolbar (default=1)

To respond to the request for a new window, the callback function should open the requested URL as appropriate, for example, in a newly created [Section 6.307](#) object.

EXAMPLE

```
'h'[]WC 'HTMLRenderer'
'h'[]WS ('Event' 'DoPopUp' 'DoPopUpCB')
```

```
html← '<a href="https://www.dyalog.com"'
html,← target="_blank">Dyalog Website</a>'
```

```
h.HTML←html
```

```
▽ DoPopUpCB msg;props;posn
[1] props←c 'URL' (3>msg)
[2] (posn size)←2↑4>msg
[3] props,←('Posn'posn)('Size'size)
[4] 'popup'[]WC'HTMLRenderer'props
▽
```

EXTENDED EXAMPLE

```

    ▽ {r}←DoPopupDemo args;html;h;c;s;e;p;d
[1]   →EndHTML
[2]   StartHTML:
[3]   A<html>
[4]   A  <head><title>DoPopupDemo</title></head>
[5]   A<body>
[6]   A  {}
[7]   A  <button type="button"
onclick="window.open('callback','_blank','width=400,height=200')">Click
me!</button>
[8]   A</body>
[9]   A</html>
[10]  EndHTML:
[11]  html←ε'A'(1↪1)“(1+StartHTML)↓EndHTML↑␣NR␣SI
[12]  :If 0∈args
[13]      renderers←0
[14]      html←'{}'␣R'This is the original window'←html
[15]      s←'Size'(0 0)
[16]      p←'Posn'(0 0)
[17]      d←'Data' 0
[18]      →Create
[19]  :Else
[20]      :Select 2>args
[21]      :Case 'DoPopup'
[22]          d←'Data'(1+[/renderers.Data)
[23]          html←'{}'␣R('This is window ',⌘2>d)←html
[24]          s←'Size'(2>4>args)
[25]          p←'Posn'(100+(>args).Posn)
[26]          →Create
[27]      :Case 'Close'
[28]          renderers~←>args
[29]      :EndSelect
[30]  :EndIf
[31]  →0
[32]  Create:
[33]  h←'HTML'html
[34]  c←'Coord' 'Pixel'
[35]  e←'Event'('onAll' 'DoPopupDemo')
[36]  renderers,←NEW'HTMLRenderer'(h c s p e d)
    ▽

```

The example function show above will display a new window when the button labelled Click me! is pressed. To start, type:

```
DoPopupDemo ' '
```

6.173 Draggable Property

Applies To

Description

This property determines whether or not an object may be the subject of a "drag and drop" operation.

Dragable is a single number with the value 0, 1 or 2. A value of 0 (which is the default) means that the object may not be drag/dropped. A value of 1 means that the object may be drag/dropped and that during the "drag" operation, a box representing the bounding rectangle around the object is displayed on the screen. A value of 2 means that the **outline** of the object is displayed as the object is dragged, or, if the object is an [Section 6.313](#) with a [Section 6.447](#) property containing the name of an [Section 6.309](#) object, the icon itself is displayed as it is dragged. For rectangular non-graphical objects, values of 1 and 2 are equivalent.

If an object is Draggable, the user may drag it by positioning the mouse pointer within the object, depressing the left mouse button, then moving the mouse with the button held down. During the drag operation, the mouse pointer is automatically changed to a "drag" symbol. The object is "dropped" by releasing the left mouse button. The effect depends upon where it is dropped, and on the action associated with the [Section 6.174](#) event for the object under the mouse pointer (if any).

If there is no object under the mouse pointer, the "drag and drop" operation is ignored. Otherwise, the object under the mouse pointer generates a [Section 6.174](#) event.

If the object under the mouse pointer is the parent of the object that has been "dragged and dropped", the default action is for the system to move that object to the new location within its parent. If you wish to allow your user to freely move an object within its parent [Section 6.227](#) or [Section 6.279](#), simply set its Draggable property to 1; the system will take care of the rest. If you want to allow the user to move an object, but you want to know about it when it happens, you can associate a callback function to the [Section 6.174](#) event that queries the new position. To permit the operation to complete, the callback function should either not return a result or it should return something other than a scalar 0. To selectively disable movement, your callback

function should return a scalar 0 in circumstances when the "drop" is not to be permitted.

If the object under the mouse pointer is not the parent of the object being dragged, the default action is for the system to ignore the operation. However, by enabling the [Section 6.174](#) event, your application can of course take whatever action is appropriate, including perhaps moving the dragged object to a new parent.

Note that a Draggable object does not generate a [Section 6.123](#) (31) event when it is dragged and dropped.

6.174 DragDrop Event 11

Applies To

Description

If enabled, this event is reported when the user drops one object (the *dragged object*) onto another (the *target object*). The event is generated by the target object, that is, the object which is being dropped **on**.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 7-element vector as follows :

[1]	Target Object	ref or character vector
[2]	Event	'DragDrop' or 11
[3]	Dragged Object	ref or character vector
[4]	Y	y-position of mouse pointer in the target object
[5]	X	x-position of mouse pointer in the target object
[6]	H	height of dragged object in the coordinate system of the target object
[7]	W	width of dragged object in the coordinate system of the target object
[8]	Shift State	numeric scalar containing the sum of the values associated with the Shift(1), Ctrl(2) and Alt(4) keys when the object was dropped.

Note that Y, H, X and W are all reported relative to the target object.

6.175 DragItems Property

Applies To: Section 6.348

Description

The DragItems property is Boolean and specifies whether or not the items in a Section 6.348 object may be drag/dropped by the user. Its default value is 1.

6.176 DrawMode Property

Applies To: Section 6.94, Section 6.190, Section 6.365, Section 6.450, Section 6.472, Section 6.605

Description

The DrawMode property provides direct control over the low-level drawing operation performed by graphical objects.

The DrawMode property specifies the current foreground mix mode. The Windows GDI uses the foreground mix mode to combine pens and interiors of filled objects with the colours already on the screen. The foreground mix mode defines how colours from the brush or pen and the colours in the existing image are to be combined.

DrawMode affects every drawing operation performed by Dyalog APL and not just the initial drawing operation when the object is created. Many of the drawing modes are additive (the result depends not just on what is being drawn, but on what is already there) and the effects may therefore vary. For this reason, DrawMode should normally be used only with un-named graphical objects.

You could use DrawMode to move or animate graphical objects in circumstances where the standard Dyalog APL behaviour was not ideal.

DrawMode is an integer with one of the following values:

Value	Name	Resulting Pixel Colour
1	R2_BLACK	Pixel is always 0.
2	R2_NOTMERGEPEN	Pixel is the inverse of the R2_MERGEPEN colour.
3	R2_MASKNOTPEN	Pixel is a combination of the colours common to both the screen and the inverse of the pen.
4	R2_NOTCOPYPEN	Pixel is the inverse of the pen colour.
5	R2_MASKPENNOT	Pixel is a combination of the colours common to both the pen and the inverse of the screen.
6	R2_NOT	Pixel is the inverse of the screen colour.
7	R2_XORPEN	Pixel is a combination of the colours in the pen and in the screen, but not in both.
8	R2_NOTMASKPEN	Pixel is the inverse of the R2_MASKPEN colour.
9	R2_MASKPEN	Pixel is a combination of the colours common to both the pen and the screen.
10	R2_NOTXORPEN	Pixel is the inverse of the R2_XORPEN colour.
11	R2_NOP	Pixel remains unchanged.
12	R2_MERGENOTPEN	Pixel is a combination of the screen colour and the inverse of the pen colour.
13	R2_COPYPEN	Pixel is the pen colour.
14	R2_MERGEPENNOT	Pixel is a combination of the pen colour and the inverse of the screen colour.
15	R2_MERGEPEN	Pixel is a combination of the pen colour and the screen colour.
16	R2_WHITE	Pixel is always 1.

6.177 DropDown Event 45

Applies To

Description

If enabled, this event is reported when the user clicks the drop-down button in a [Section 6.121](#), [Section 6.122](#), [Section 6.108](#), [Section 6.143](#) or [Section 6.382](#) object, just before the drop-down list, colour selection, calendar or -menu is displayed.

For a [Section 6.55](#) this event only applies if the Style of the [Section 6.55](#) is `Split`. For such a [Section 6.55](#) and for the [Section 6.56](#) object there is no default processing for the event and it is the responsibility of the programmer to take appropriate action in a call-back function.

For a [Section 6.143](#) this event only applies if the Style of the [Section 6.143](#) is `'Combo'`.

The event message reported as the result of `□□DQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DropDown' or 45

This event is reported for information only and cannot be disabled or modified in any way.

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

6.178 DropFiles Event 450

Applies To

Description

If enabled, this event is reported when the user drags a file icon or a set of file icons and drops them onto the object. The system takes no action other than to report the event.

Note that the event is only reported if [Section 6.3](#) is set to 1.

The event message reported as the result of `□□DQ`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DropFiles' or 450
[3]	Files	Vector of character vectors containing the file names.
[4]	Item number	Integer. The index of the item within the object onto which the file(s) was dropped. Applies only to objects that have an <u>Items</u> property such as List, <u>ListView</u> and <u>TreeView</u> .
[5]	Shift state	Integer. Sum of 1=shift key, 2=Ctrl key, 4=Alt key

6.179 DropObjects Event 455

Applies To

Description

If enabled, this event is reported when the user drags an object icon or a set of object icons from the *Explorer Tool* or *Find Objects Tool* (which are part of the Dyalog APL Session) and drops them onto the object. The system takes no action other than to report the event. You can use this event to extend drag-drop functionality in your Session. For example, you could perform an operation by drag-dropping an APL object icon onto a Button in the Session toolbar.

The event message reported as the result of □DQ, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'DropObjects' or 455
[3]	Objects	Vector of character vectors containing the object names.
[4]	Item number	Integer. The index of the item within the object onto which the file(s) was dropped. Applies only to objects that have an <u>Items</u> property such as List, <u>ListView</u> and <u>TreeView</u> . Otherwise this value is -1.
[5]	Shift state	Integer. Sum of 1=Shift key, 2=Ctrl key, 4=Alt key

6.180 Duplex Property

Applies To: [Section 6.455](#)

Description

Specifies whether pages are printed on separate sheets or back-to-back.

Duplex is a character vector which is either empty or contains 'Simplex', 'Vertical', or 'Horizontal'.

The default value for Duplex is derived from the current printer setting and 'Vertical' and 'Horizontal' are only effective if the printer supports a duplex capability.

6.181 DuplicateColumn Method 178

Applies To: [Section 6.263](#)

Description

This method is used to duplicate a column in a [Section 6.263](#) object.

The argument to DuplicateColumn is a 2, 3, 4 or 5-element vector as follows:

[1]	Source Column number	number of the column (integer) to be duplicated
[2]	Target Column number	new column number (integer)
[3]	Comment flag	0 or 1 (optional, default 1)
[4]	Lock flag	0 or 1 (optional, default 1)
[5]	Undo flag	0 or 1 (optional; default 0)

If the *Comment flag* is 1 (the default), any Comments associated with cells in the source column are duplicated in the target column.

If the *Lock flag* is 1 (the default), the lock state of the column is duplicated; otherwise, the new column is not locked.

If the *Undo flag* is 1, the column may subsequently be restored by invoking the Undo method. If this element is omitted or is 0, the operation may not be undone.

6.182 DuplicateRow Method 177

Applies To: [Section 6.263](#)

Description

This method is used to duplicate a row in a [Section 6.263](#) object.

The argument to DuplicateRow is a 2, 3, 4 or 5-element vector as follows:

[1]	Source Row number	number of the row (integer) to be duplicated
[2]	Target Row number	new row number (integer)
[3]	Comment flag	0 or 1 (optional, default 1)
[4]	Lock flag	0 or 1 (optional, default 1)
[5]	Undo flag	0 or 1 (optional; default 0)

If the *Comment flag* is 1 (the default), any Comments associated with cells in the source row are duplicated in the target row.

If the *Lock flag* is 1 (the default), the lock state of the row is duplicated; otherwise, the new row is not locked.

If the *Undo flag* is 1, the row may subsequently be restored by invoking the Undo method. If this element is omitted or is 0, the operation may not be undone.

6.183 DyalogCustomMessage1 Event 95

Applies To: [Section 6.227](#)

Description

This event allows external applications and dynamic link libraries to insert events into the Dyalog APL/W message queue.

DyalogCustomMessage1 may be invoked from a C program as follows:

```
msg=RegisterWindowMessage( "DyalogCustomMessage1" );
SendMessage( hWnd,msg,wParam,lParam );
```

where hWnd is the window handle of the object in the Dyalog APL Workspace. If the object is a [Section 6.227](#), this may be obtained using FindWindow(). If not, hWnd may be passed to the external process as an argument to a function.

The parameters wParam and lParam are reported as numeric arguments to the APL callback function.

NOTE: It is not possible to pass pointers to data in wParam or lParam. When the APL callback executes the pointers may not be valid.

If a callback function is attached to the event, the callback function will be run when the event reaches the top of the queue.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to -1 or by returning 0 from a callback function.

The result of a callback function is **not** returned to the external application.

The event message reported as the result of □DQ, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Dy a l o g C u s t o m M e s s a g e 1 ' or 95
[3]	wparam	integer
[4]	lparam	integer

6.184 EdgeStyle Property

Applies To

Description

This property is used to give a 3-dimensional appearance to screen objects.

This is achieved by drawing the object with a grey or white background colour and by drawing a border around it using various combinations of black, white and dark grey lines. Note that this border is drawn *outside* a control but *inside* a [Section 6.227](#) or [Section 6.578](#).

Note that EdgeStyle is not honoured for objects which have a natural built-in 3-dimensional appearance.

The value of the EdgeStyle property is a character vector chosen from the following :

'None'	Object is drawn with no 3-dimensional effects and the EdgeStyle properties of its children are ignored (treated as None).
'Plinth'	Object is drawn with a light shadow along its top and left edges and a dark shadow along its bottom and right edges. This gives the illusion of a raised effect.
'Recess'	Object is drawn with a dark shadow along its top and left edges and a light shadow along its bottom and right edges. This gives the illusion of a sunken effect.
'Groove'	Object is drawn with a border that has the appearance of a groove.
'Ridge'	Object is drawn with a border that has the appearance of a ridge.
'Shadow'	Object is drawn with a dark border line along its top and left edges.
'Default'	Object itself is drawn with no 3-dimensional border, but the values of the EdgeStyle properties of its children are observed.
'Dialog'	Used in conjunction with ('Border' 2), this gives a Form the appearance of a standard 3-dimensional dialog box. This setting applies only to a <u>Form</u> or a <u>SubForm</u>

For the [Section 6.489](#) object, the EdgeStyle property may be 'None' or 'Default'. If EdgeStyle is 'None', screen objects are drawn without 3-dimensional effects of any kind and the value of their EdgeStyle property is ignored. If EdgeStyle is 'Default', all controls are drawn using their default EdgeStyle properties.

Note that [Section 6.399](#), [Section 6.208](#) and the set-up dialog box associated with the [Section 6.455](#) object are all drawn with 3-dimensional effects regardless of the value of EdgeStyle on [Section 6.489](#). These objects do not have their own EdgeStyle properties.

If you set EdgeStyle to 'None' on the [Section 6.489](#) object, all your objects will (by default) be drawn without 3-dimensional effects.

6.185 Edit Object

Purpose: Allows user to enter or edit data.

Parents ActiveXControl CoolBand Form Grid Group PropertyPage SubForm ToolBar ToolControl

Children Circle Cursor Ellipse Font Marker Poly Rect Text Timer

Properties Type Text Posn Size Style Coord Align Border Justify Active Visible Event VScroll HScroll SelText Sizeable Dragable FontObj FCol BCol CursorObj AutoConf Data Attach TextSize EdgeStyle Handle Hint HintObj Tip TipObj FieldType MaxLength Decimals Password ValidIfEmpty ReadOnly FormatString Changed Value Translate Accelerator AcceptFiles WantsReturn KeepOnClose Redraw TabIndex Cue ShowCueWhenFocused HasClearButton MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ChooseFont

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDblClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap Select BadValue Change KeyError

Description

The value of the [Section 6.577](#) property, which may be 'Single' or 'Multi', determines whether the object presents a single-line data entry field or an area for viewing and editing a large block of text.

Single-Line Edit

The [Section 6.206](#) property (which applies only to a single-line Edit object) is either an empty vector (the default) or specifies the type of the field as reflected by its [Section 6.641](#) property. If [Section 6.206](#) is empty, the [Section 6.641](#) of the field may be a number or a character vector, according to the contents defined by its [Section 6.606](#) property (this is always a character vector).

The [Section 6.13](#) property, which applies only to single-line Edit objects, specifies the vertical alignment of the text. It may be 'None' (the default), 'Centre' or 'Center'. [Section 6.13](#) may only be set when the object is created and may not subsequently be changed.

If [Section 6.206](#) is 'Char' the [Section 6.641](#) of the field is forced to be a character vector, even if the contents defined by its [Section 6.606](#) property is entirely numeric.

If [Section 6.206](#) is 'Numeric', 'LongNumeric', 'Currency', 'Date', 'LongDate', or 'Time' the [Section 6.641](#) contains a number. For fields of these types, basic validation is provided during user input. The field is revalidated when the user attempts to "leave" it and at this point the object will generate a [Section 6.38](#) event if its contents are inconsistent with its [Section 6.206](#).

Note that when an Edit object is used as the [Section 6.321](#) property of a [Section 6.263](#), it is the [Section 6.641](#) of the Edit object (and not its [Section 6.606](#) property) that is used to update the [Section 6.642](#) property (that is, the contents of) the [Section 6.263](#) when the user moves away from the cell.

The [Section 6.371](#) property defines the maximum number of characters that the user may enter into the object.

The [Section 6.639](#) property may be 0 (the default) or 1 and specifies whether or not the Edit field is deemed to be valid if it is empty.

The [Section 6.445](#) property specifies the character that is displayed in response to the user typing a character. Normally, [Section 6.445](#) is empty (the default) and the object displays the character that was entered. However, if you set [Section 6.445](#) to (say) an asterisk (*) this symbol will be displayed instead of the characters the user has entered. Note however that the [Section 6.606](#) and [Section 6.641](#) properties will reflect what the user typed.

The [Section 6.303](#) property determines whether or not the data may be scrolled. If [Section 6.303](#) is 0, the data is not scrollable, and the user cannot enter more characters once the field is full. If [Section 6.303](#) is -1 or -2 the field is scrollable, and there is no limit on the number of characters that can be entered. In neither case however is a horizontal scrollbar provided. [Section 6.303](#) may only be set when the object is created and may not subsequently be changed.

Multi-Line Edit

If the [Section 6.577](#) is 'Multi', [Section 6.606](#) may set using a simple character vector, a simple matrix, or a vector of vectors. If [Section 6.606](#) is specified by a matrix or by a vector of vectors, "new-line" characters are automatically added at the end of each line in the Edit control.

The user may insert a "new-line" character in the text by pressing Ctrl-Enter. If [Section 6.606](#) was set by a matrix, it is returned as a matrix. Otherwise it is returned as a vector of vectors. "New-line" characters are not returned. If [Section 6.606](#) was not specified by *Dyalog APL Language: Wc* or *Dyalog APL Language: Ws* it is returned an empty matrix (1 0p ' '). However, if [Section 6.606](#) was not specified, but the user types and then empties the field, it is returned as an empty nested array (, c ' ')

The [Section 6.336](#) property determines whether the text in a multi-line Edit object is 'Left', 'Right', or 'Centre' justified. Setting [Section 6.336](#) to 'Centre' or 'Right' also forces word-wrapping and disables horizontal scrolling, whatever the value of

Section 6.303. Note that the keyword 'Centre' may also be spelled 'Center'. Section 6.336 may only be specified when the object is created using `⎕WC`.

If Section 6.336 is 'Left', the Section 6.303 property determines whether or not text may be scrolled horizontally. If Section 6.303 is set to `⌵2`, each individual line may be any length, but the object does not have a horizontal scrollbar. Sideways scrolling is achieved using the cursor keys, or by typing. If Section 6.303 is `⌵1`, each individual line may be of any length and the object will have a horizontal scrollbar. If Section 6.303 is `0`, lines are automatically "word-wrapped" at the right edge of the object. This means that the number of lines displayed may be greater than the number of lines implied by the rows of the matrix or the number of vectors supplied. In particular, if you specify a single long vector, it will be broken up into lines for you on the display, but still returned as a single vector by *Dyalog APL Language: Wg*.

The Section 6.646 property determines whether or not data may be scrolled vertically and whether or not the object has a vertical scrollbar. A value of `0` inhibits scrolling; `⌵2` means scrollable, without a scrollbar; `⌵1` means scrollable with a scrollbar. Section 6.646 may only be set when the object is created and may not subsequently be changed.

The setting of Section 6.336 forces word-wrapping.

The Section 6.521 property identifies the portion of the text that is selected. It may be used to pre-select (and highlight) a part of the text, or to report the part of the text selected by the user. Section 6.521 is a 2-element integer vector which specifies the start and end of the selected area. Its structure depends upon the nature of the data specified by Section 6.606. See the description of Section 6.521 for details.

If the user changes any data in the field **and** attempts to change focus to another object, the Edit object will generate a Section 6.85 event. You can use this to validate the new data in the field.

Note

For full functionality (in particular, for the Section 6.132 property to apply), the Edit object requires that Section 8.6

is enabled.

6.186 EditImage Property

Applies To: Section 6.122

Description

Specifies whether or not the edit control portion of the ComboEx displays an image for selected items.

EditImage is a single number with the value 0 or 1 (the default). If EditImage is 1, the image associated with the selected item is displayed in the edit control, portion of the ComboEx object, to the left of the text. If EditImage is 0, only the item text is displayed in the edit control.

6.187 EditImageIndent Property

Applies To: [Section 6.122](#)

Description

Specifies whether or not the indents associated with items in a ComboEx object are honoured in the edit control portion of the ComboEx.

EditImageIndent is a single number with the value 0 or 1 (the default).

If EditImageIndent is 1, the selected item which is displayed in the edit control portion of the ComboEx object is indented in the same way as when it is displayed in the dropdown portion of the object. The amount of indentation is specified by the Indents property.

If EditImageIndent is 0, the item displayed in the edit control portion of the ComboEx is not indented.

6.188 EditLabels Property

Applies To: [Section 6.348](#), [Section 6.629](#)

Description

The EditLabels property is Boolean and specifies whether or not the labels (specified by the Items property) in a [Section 6.348](#) or [Section 6.629](#) object may be edited by the user. Its default value is 0 (editing is not allowed).

If EditLabels is 1, the user begins editing by clicking the label of the item that has the focus. This causes a pop-up edit box to appear around the item and allows the use to change it. A [Section 6.46](#) event is reported at the start of the edit operation and an

[Section 6.193](#) event is reported on its completion. You may control the edit of a particular label using callback functions attached to these events.

6.189 Elevated Property

Applies To: [Section 6.55](#)

Description

The [Section 6.189](#) property applies only to a [Section 6.55](#) whose Style is 'CommandLink'.

Elevated is a Boolean scalar with a default value of 0. When set to 1, the icon on the CommandLink button changes from a "green arrow" to a "shield". This is intended to convey to the user that the action associated with the [Section 6.55](#) requires *Elevation*. This is a feature of *User Account Control* in Windows 7. See your Windows documentation for further information. Note that APL does not take any action (other than to cause the icon to change) when Elevated is set to 1. This is the responsibility of the programmer.

6.190 Ellipse Object

Purpose: A Graphical object to draw ellipses, arcs, and pie-slices.

Parents ActiveXControl Animation Bitmap Button ButtonEdit Combo ComboEx Edit Form Grid Group Label List ListView MDIClient Metafile Printer ProgressBar PropertyPage PropertySheet RichEdit Scroll Spinner Static StatusBar SubForm TabBar TipField ToolBar TrackBar TreeView UpDown

Children Timer

Properties Type Points Size FCol BCol Start End ArcMode LStyle LWidth FStyle FillCol Coord Visible Event Dragable OnTop CursorObj AutoConf Data Accelerator KeepOnClose DrawMode MethodList ChildList EventList PropList

Methods Detach

Events Close Create DragDrop MouseDown MouseUp MouseMove MouseDbIClick Help Select

Description

This object duplicates much of the functionality of the [Section 6.94](#) object, but differs in two major respects. Firstly, ellipses, circles, and arcs are specified in terms of their **bounding rectangles**, rather than in terms of their centre(s) and radii. Secondly, the Ellipse object behaves like any other (rectangular) object when it is resized by its parent. The [Section 6.94](#) object behaves differently in that when resized by its parent, it maintains a constant ratio between its **physical** height and width.

The [Section 6.449](#) property specifies one or more sets of co-ordinates which define the position(s) of one or more bounding rectangles. The position is defined to be the position of the corner that is **nearest** to the origin of its parent. The default is therefore its top-left corner.

The [Section 6.556](#) property specifies the height and width of each bounding rectangle, measuring away from the origin. To obtain a perfect circle, you must take the **aspect ratio** of the device into account. This is available from the [Section 6.158](#) property of the [Section 6.489](#) and [Section 6.455](#) objects. Alternatively you can use the [Section 6.94](#) object.

The [Section 6.568](#) and/or [Section 6.192](#) properties are used to draw partial ellipses and circles. They specify start and end angles respectively, measuring from the x-axis at the centre of the bounding rectangle in a counter-clockwise direction and are expressed in radians. The type of arc is controlled by [Section 6.29](#) as follows:

ArcMode	Effect
0	An arc is drawn from Start to End .
1	An arc is drawn from Start to End . In addition, a single straight line is drawn from one end of the arc to the other, resulting in a segment.
2	An arc is drawn from Start to End . In addition, two lines are drawn from each end of the arc to the centre, resulting in a pie-slice.

[Section 6.360](#) and [Section 6.361](#) define the style and width of the lines used to draw the boundaries of the ellipse(s), circle(s) or arc(s). [Section 6.205](#) and [Section 6.45](#) determine the colour of the lines.

[Section 6.231](#) specifies whether or not the ellipse(s), circle(s) or arc(s) are filled, and if so, how. For a solid fill ([Section 6.231](#) 0), [Section 6.214](#) defines the fill colour used. For a pattern fill ([Section 6.231](#) 1-6) [Section 6.214](#) defines the colour of the hatch lines and [Section 6.45](#) the colour of the spaces between them.

The value of [Section 6.173](#) determines whether or not the object can be dragged. The value of [Section 6.35](#) determines whether or not the Ellipse object is resized when its parent is resized.

The structure of the property values is best considered separately for single and multiple ellipses, circles or arcs.

Single Ellipse, Circle or Arc

For a single ellipse, circle or arc, [Section 6.449](#) is a 2-element vector which specifies the y-coordinate and x-coordinate of the top-left corner of the bounding rectangle.

[Section 6.556](#) is also a simple 2-element vector whose elements specify the height and width of the bounding rectangle.

[Section 6.360](#) and [Section 6.361](#) are both simple scalar numbers.

[Section 6.231](#) is either a single number specifying a standard fill pattern, or the name of a [Section 6.47](#) object which is to be used to fill the ellipse, circle or arc.

[Section 6.205](#), [Section 6.45](#) and [Section 6.214](#) are each either single numbers representing standard colours, or 3-element vectors which specify colours explicitly in terms of their RGB values.

EXAMPLES

First make a [Section 6.227](#) :

```
'F' WC 'Form'
```

Draw a complete ellipse within the bounding rectangle located at (y=10, x=5) with (height=30, width=50) :

```
'F.E1' WC 'Ellipse' (10 5)(30 50)
```

Draw an elliptical arc within the same bounding rectangle as above, occupying the upper right quadrant (0 to 90 degrees):

```
'F.E1' WC 'Ellipse' (10 5)(30 50)('End'(00.5))
```

Ditto, but between 45 and 135 degrees :

```
'F.E1' WC 'Ellipse' (10 5)(30 50)('Start'(00.25))('End'(00.75))
```

Ditto, but join the points of the arc to the centre of the ellipse, making a "pie-slice":

```
'F.E1' DW 'Ellipse' (10 5)(30 50)('Start'(00.25))('End'(00.75))
('ArcMode' 2)
```

Ditto, but use a green line and solid red fill :

```
'F.E1' DW 'Ellipse' (10 5)(30 50)('Start'(00.25))('End'(00.75))
('ArcMode' 2)('FCol' 0 0 255)('FStyle' 0)('FillCol' 255 0 0)
```

Multiple Ellipses, Circles or Arcs

To draw a set of ellipses, circles, or arcs with a single name, [Section 6.449](#) may be a simple 2-element vector (specifying the location of all the bounding rectangles), **or** a 2-column matrix whose first column specifies their y-coordinates and whose second column specifies their x-coordinates, **or** a 2-element nested vector whose first element specifies their y-coordinate(s) and whose second element specifies their x-coordinate(s).

Likewise, [Section 6.556](#) may be a simple 2-element vector (applying to all the bounding rectangles), **or** a 2-column matrix whose first column specifies their heights and whose second column specifies their widths, **or** a 2-element nested vector whose first element specifies their height(s) and whose second element specifies their width(s).

If specified, [Section 6.568](#) and/or [Section 6.192](#) define arcs in terms of the angles made by drawing a line from the centre of the bounding box to the two ends of the arc. Both properties may be simple scalars, or vectors containing one element per arc drawn.

If [Section 6.568](#) is specified, but not [Section 6.192](#), end angles default to (-1++\Start), 02. If [Section 6.192](#) is specified, but not [Section 6.568](#), start angles default to 0, -1++\End

This means that you can draw a pie-chart using either [Section 6.568](#) or [Section 6.192](#) angles; you do not have to specify both.

[Section 6.29](#), [Section 6.360](#) and [Section 6.361](#) may each be simple scalar values (applying to all the ellipses, circles or arcs) or simple vectors whose elements refer to each of the corresponding ellipses, circles or arcs in turn.

[Section 6.231](#) may be a simple scalar numeric or a simple character vector ([Section 6.47](#) name) applying to all rectangles, or a vector whose elements refer to each of the corresponding ellipses, circles or arcs in turn.

Similarly, [Section 6.205](#), [Section 6.45](#) and [Section 6.214](#) may each be single numbers or a single (enclosed) 3-element vector applying to all the rectangles. Alternatively, these properties may contain vectors whose elements refer to each of the rectangles in turn.

If so, their elements may be single numbers or nested RGB triplets, or a combination of the two.

The [Section 6.128](#), [Section 6.173](#) and [Section 6.140](#) properties are specified for the object as a whole, and may not be allocated different values for each individual ellipse, circle or arc that is drawn.

EXAMPLES

First make a [Section 6.227](#) :

```
'F' □WC 'Form'
```

Draw two ellipses in bounding rectangles located at (y=5, x=10) and (y=5, x=60), each of (height=40, width=10)

```
'F.E1' □WC 'Ellipse' ((5 5)(10 60)) (40 10)
```

Ditto, using scalar extension for (y=5) :

```
'F.E1' □WC 'Ellipse' (5(10 60)) (40 10)
```

Ditto, but draw the first with (height=40, width=30) and the second with (height=20, width=10) :

```
'F.E1' □WC 'Ellipse' (5(10 60)) ((40 20)(30 10))
```

Draw an elliptical Pie-Chart in a bounding rectangle located at (y=5, x=10) with a height and width equal to 40% of the height and width of the parent [Section 6.227](#). Each of the 4 pie-slices is bounded by a black line :

```
Data ←12 27 21 40
ANGLES←0, -1+((02)÷+/Data)×+\Data
COLS←(255 0 0)(0 255 0)(255 255 0)(0 0 255)
PATS←1 2 3 4

'F.PIE' □WC 'Ellipse'(5 10)(40 40)
      ('Start' ANGLES)('ArcMode' 2)
      ('FCol' (c0 0 0))('FStyle' PATS)
      ('FillCol' COLS)
```

6.191 Encoding Property

Applies To: [Section 6.604](#)

Description

The Encoding property is a character vector that specifies how character data are encoded or translated.

The possible values are 'None', 'UTF-8', 'Classic', or 'Unicode', depending upon the value of the [Section 6.577](#) property.

Table: Unicode Edition

Style	Encoding	Description
'Raw'	'None'	Not applicable. Only integer data may be transmitted/received.
'Char'	'None'	Transmission is limited to characters with Unicode code points in the range 0-255. Attempting to transmit (or receive) a character outside this range will cause DOMAIN ERROR.
	'UTF-8'	Characters are transmitted/received using the UTF-8 encoding scheme.
'APL'	'Classic'	Characters are transmitted/received as indices of □AV, and translated according to the current value of □AVU . An attempt to transmit or receive a characters not present in □AVU will cause TRANSLATION ERROR
	'Unicode'	Characters are transmitted/received <i>as is</i> (as Unicode code points).

Table: Classic Edition

Style	Encoding	Description
'Raw'	'None'	Not applicable. Only integer data may be transmitted/received.
'Char'	'None'	Characters (which are represented internally as indices of <code>AV</code>) are translated to and from ASCII using the Output Translate Table <code>win.dot</code> .
	'UTF-8'	Characters are converted to/from Unicode using <code>AVU</code> and transmitted/received using the UTF-8 encoding scheme. An attempt to transmit or receive a characters not present in <code>AVU</code> will cause <code>TRANSLATION ERROR</code> .
'Raw'	'Classic'	Characters are transmitted/received as indices of <code>AV</code> .
	'Unicode'	Characters are converted to/from Unicode using <code>AVU</code> and transmitted/received as Unicode code points. An attempt to transmit or receive a characters not present in <code>AVU</code> will cause <code>TRANSLATION ERROR</code> .

The default value of Encoding depends upon the value of [Section 6.577](#) as indicated. Default values are highlighted thus in the above tables.

An attempt to set the value of Encoding to a value not valid for the current [Section 6.577](#), as implied by the above tables, will cause `DOMAIN ERROR`.

If you change the value of the [Section 6.577](#) property, the value of Encoding will remain unchanged if it is valid for the new Style. Otherwise it will revert to the default value for the new value of [Section 6.577](#).

```

's0'WC'TCPSocket' ('LocalPort' 2001)
s0.(Style Encoding)
Char  None

s0.Style←'APL'
s0.(Style Encoding)
Apl  Classic

```

6.192 End Property

Applies To: [Section 6.94](#), [Section 6.190](#)

Description

This property specifies one or more end-angles for an arc, pie-slice, or chord of a circle or ellipse. It may be used in conjunction with [Section 6.568](#) which specifies start angles. Angles are measured counter-clockwise from the x-axis at the centre of the object.

If a single arc is being drawn, End is a single number that specifies the end angle of the arc in radians ($0 \leq \text{End} < 2\pi$). If multiple arcs are being drawn, End is either a single number as before (the end angle for several concentric arcs) or a numeric vector with one element per arc.

If [Section 6.568](#) is not specified, the default value of End is 2π . Otherwise, the default value of End is $(-1+\text{Start}), 2\pi$.

6.193 EndEditLabel Event 301

Applies To: [Section 6.348](#), [Section 6.629](#)

Description

If enabled, this event is reported when the user signals completion of an edit operation in a [Section 6.348](#) or [Section 6.629](#) object. This occurs when the item being edited loses the focus or when the user presses the Enter key. The default processing for the event is to update the item label (string) with the edited text in the pop-up edit box.

You may disable the update operation by setting the action code for the event to -1. You may also prevent the update from occurring by returning 0 from a callback function. You may specify the text used to update the item by returning the event message (containing the desired text) from a callback function. Finally, you may change the text of any item dynamically by calling EndEditLabel as a method.

The event message reported as the result of `DDQ`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'EndEditLabel' or 301
[3]	Item number	Integer. The index of the item.
[4]	Text	character vector containing the text that will be used to update the item's label.

6.194 EndSplit Event 282

Applies To: [Section 6.566](#)

Description

If enabled, this event is reported when the user releases the left mouse button to signify the end of a drag operation on a [Section 6.566](#) object.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to -1 or by returning 0 from a callback function.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'EndSplit' or 282
[3]	Y	y-position of top left corner
[4]	X	x-position of top left corner
[5]	H	height of the Splitter
[6]	W	width of the Splitter

See also [Section 6.570](#), [Section 6.567](#).

6.195 EnterReadOnlyCells Property

Applies To: [Section 6.263](#)

Description

This is a Boolean property that specifies whether or not the user may visit read-only cells in a [Section 6.263](#) object. Its default value is 1.

In this context, a *read-only* cell is one that satisfies one or more of the following conditions:

- it has no associated [Section 6.321](#) object
- its associated Input object is a [Section 6.341](#)
- its associated Input object is an [Section 6.185](#) object with [Section 6.470](#) set to 1
- its associated Input object is inactive ([Section 6.5 0](#))

If `EnterReadOnlyCells` is set to 0 and the user clicks the mouse on a *read-only* cell, the current cell does not change although [Section 6.73](#), [Section 6.83](#) and [Section 6.72](#) events are reported if enabled. If the user presses a cursor movement key that would otherwise cause the cursor to move into a *read-only* cell, the cursor moves instead to the nearest *editable* cell in the appropriate direction.

6.196 Event Property

Applies To

Description

This property defines how an object responds to user actions.

Unlike other properties which only have a single value, this property has a value corresponding to each of the different types of event that may be generated by a particular object. Consequently the syntax for setting the Event property differs from the general syntax that applies to other properties.

Two syntactic forms are allowed:

- A 3 or 4-item vector containing the property name `'Event'`, followed by the Event Type(s), a value which determines the action to be taken, and an optional array that will be supplied as a left argument to the callback function
- A composite vector whose first element contains the property name `'Event'`, followed by a series of 2 or 3-element vectors, each defining the action to be taken for a different Event Type (or types).

EXAMPLES

```
'Event' 'MouseUp' 'foo' 88
'Event' ('MouseUp' 'MouseDown') 'foo' 88
'Event' ('MouseUp' 'foo' 88)('MouseDown' 'goo')
```

Like any other property, the Event property can be set using assignment. However, certain special considerations apply which are discussed later.

When you specify the Event property using `⌈WC` or `⌈WS`, the action to be taken for an event type or types is specified by a 2 or 3-element vector containing:

Element	Item	Description
[1]	Type(s)	see below
[2]	Action	numeric scalar or character vector -1 inhibit (ignore) event 0 handle event, do not report to APL 1 handle event, then report to APL f n name of callback function f n& name of callback function to be executed <i>asynchronously</i> #expr expression to be executed
[3]	Arg	any array (optional)

Event Types

The first element, *Type(s)* may be one of the following:

- A character vector containing an event name (for example 'MouseUp')
- A numeric scalar containing an event number (for example 2). If the number is not one of the built-in event numbers generated by the object, it is assumed to be a user-defined event which can (only) be generated anagrammatically using `□NQ`
- A vector of character vectors containing a list of event names, for example ('MouseDown' 'MouseUp'). This may be used as a shortcut to associate several different types of events with the same action
- An Event name preceded by the string 'on' (for example 'onMouseUp')
- An event number preceded by the string 'on' (for example 'on99'). This syntax is intended for user-defined events although it can be used with regular events too.

The *onEvent* syntax causes all objects reported in the *event message* (see below) to be identified by a *ref*. Otherwise, objects reported in the event message are identified by name.

Action

Inhibit (-1)

If *Action* is set to -1, the event is inhibited (if possible) by APL. If, for example, you set the action on a KeyPress event to -1, all keystrokes for the object in question will be ignored. Similarly, if you set the *action* on a Close event for a Form to -1, the user will

be unable to close the Form. This is possible because APL intercepts most events before Windows itself takes any action. However, certain events (for example, focus change events) are not notified to APL until **after** the event has occurred and **after** Windows has itself responded in some way. In these circumstances it is not always practical for APL to undo what Windows has already done, and an action code of -1 is treated as if it were 0. For further details, see the individual entries for each event type in this chapter.

Default Processing (0)

If *Action* is set to 0 (the default), the event is processed by APL and Windows in the normal way (this is referred to herein as the *default processing*) but your program is not notified in any way that the event has occurred. For example, the default processing for a keystroke is to action it and either echo a character in the object or perform some other appropriate function.

Terminate □DQ (1)

If *Action* is set to 1, the event is first processed by APL (and Windows) in the normal way, then □DQ terminates, returning an *event message* as its result. The format of the event message is given under the description of each event type.

Callback (function name)

If *Action* is set to a character vector that specifies the name of a function, this function (termed a *callback*) will be executed automatically by □DQ every time the event occurs. The function may be a traditional defined function or a dfn.

A traditional defined function may be monadic, dyadic, or niladic. If dyadic, the left argument may be optional. A niladic callback may be appropriate if the function can perform its task without needing to interrogate the event message.

Unless the callback function is niladic, it will be supplied a right argument (ω for a dfn) containing the *event message* and a left argument (α for a dfn) of the value of the array *Arg* (if specified).

The function may be defined to return no result, a result, or a shy result. The result determines how the event is handled.

The *default processing* of the event is deferred until **after** the callback has been run, and may be inhibited or modified by its result. If the callback function returns no result,

or returns a scalar 1, normal processing of the event is allowed to continue as soon as the callback completes. If the callback returns a scalar 0, normal processing of the event is inhibited and the effect is identical to setting *Action* to `-1`. A callback function may also return an event message as its result. If so, `□DQ` will action *this* event rather than the original one that fired the callback.

If a callback function does not exist at the instant it is invoked, `□DQ` terminates with a `VALUE ERROR`. However, the name of the missing function is reported in the Status Window.

Asynchronous Callback (function name followed by &)

If *Action* is set to a character vector that specifies the name of a callback function, followed by the character `&`, the callback function will be executed asynchronously in a new thread when the event occurs.

For example, the event specification:

```
'Event' 'onSelect' 'DoIt&'
```

tells `□DQ` to execute the callback function `DoIt` asynchronously as a thread when a `Select` event occurs on the object. Note that a callback function executed in this way should not return a result (because `□DQ` does not wait for it) and any result will be displayed in the Session window.

Execute

If action code is set to a character vector whose first element is the execute symbol (`⍎`) the remaining string will be executed automatically whenever the event occurs. The default processing for the event is performed first and may not be changed or inhibited in any way.

Notice that when you specify the action to be taken on the occurrence of an event there is a great difference between `'FOO'` and `'⍎FOO'`. The former causes APL to invoke the function `FOO` as a *callback function*. If the function takes an argument, APL will supply it with the event message. Secondly, the result (if any) of the function `FOO` will be used by APL and may cause the event to be disabled or changed in some way. In the second case, APL will perform the default processing for the event and then execute `FOO` without supplying an argument. If the function returns a result, it will be displayed in the Session.

Optional Left Argument (Arg)

If specified, *Arg* is an array whose value will be passed as the left argument to a callback function when that particular event (or events) is generated. Note that this is a constant defined when the value is assigned to the Event property.

If the callback function is defined to take an explicit left argument and *Arg* was not specified, the call will fail with the error message:

```
SYNTAX ERROR: The function requires a left argument
```

If the callback function is defined to take an optional left argument and *Arg* was not specified, a reference to the left argument (α for a dfn) will generate `VALUE ERROR`.

Event Message

When a callback function is invoked by `□DQ`, the corresponding event message is supplied as its right argument. The event message is a vector whose first 2 elements identify the object that generated the event and the type of the event. Additional elements may be provided, depending upon the type of the event.

The same event message is returned as a (shy) result by `□DQ` when it is terminated by an event whose Action is set to 1.

Object(s)

The first element of the event message always identifies the object that generated the event. Other elements may identify other objects associated with the event. For example, a DragDrop event reports both the object being dropped, and the object on which it is being dropped.

Objects are identified by *names* or *refs*. If the Event property was set using the *onEvent* syntax (whereby the event name or number is prefixed by the string 'on'), for example, 'onSelect' or 'on99', objects are identified by *refs*. This is also true if the object which generated the event has no name (that is, was created by `□NEW`). Otherwise, objects are identified by their names.

Event Type

If, when the event type was specified it was identified by its name, the second element of the event message will be a character vector containing that name. If it was

identified by its number, the second element of the event message will be an integer containing that number. If the event type was identified using the *onEvent* syntax, the second element of the event message will be a character vector containing the prefix 'on' followed by the event name, even if it had been specified by number. The exception is that if the event is a user-defined event, the second element of the event message will be a character vector containing the prefix 'on' followed by the character representation of the user-defined event number.

Specifying the Event property using Assignment

There are two ways to specify the Event property using assignment; you can specify the entire set of events, or you can set events one by one (see below).

To specify the entire set of events, you assign an array to the Event property. The array must contain one or more nested vectors, each containing 2 or 3 elements (*Type*, *Action* and optionally *Arg*) as described above.

Examples (F1 is a Form)

```
F1.Event ← 'onMouseDown' 'FOO'
```

Means: invoke callback function F00 on MouseDown, the first element of the right argument to F00 will contain a *namespace reference* to F1. All other events perform their default actions.

```
F1.Event ← 'MouseDown' 'F00'
```

Means: invoke callback function F00 on MouseDown, the first element of the right argument to F00 will contain the *character vector* 'F1'. All other events perform their default actions.

```
F1.Event ← ('onMouseDown' 'F00') ('onMouseUp' 'F00')
```

Means: invoke callback function F00 on MouseDown and MouseUp. All other events perform their default actions.

```
F1.Event, ← ← 'onMouseMove' 'F00' ('THIS' 1)
```

Means: add a callback function F00 on the MouseMove event. The function will receive the array ('THIS' 1) as its left argument. All other events perform their default actions.

Specifying Individual Event types using Assignment

To define the action to be taken for individual events, one by one, you use the *onEvent* syntax and make the assignment to the event name prefixed by the string 'on'.

EXAMPLES

```
F1.onMouseDown ← 'FOO'
```

Means: invoke callback function F00 on MouseDown.

```
F1.onMouseUp ← 'FOO'
```

Means: add the same callback for MouseUp.

```
F1.onMouseMove ← 'FOO' ( 'THIS' 1 )
```

Means: add the same callback function F00 for the MouseMove event. The function will receive the array ('THIS' 1) as its left-argument.

Notice that you must use the 'on' prefix; you cannot assign to the Event name itself. This would cause an error:

```
F1.MouseUp←'foo'
SYNTAX ERROR: Invalid modified assignment, or an attempt was made to change nameclass on assignment
F1.MouseUp←'foo'
^
```

Specifying the Event property using □WC and □WS

When you set the Event property using □WC and □WS you define the actions for the event types that you specify in the argument, leaving the actions for all other event types unchanged. When you create an object with □WC, all unspecified event types will be unhandled; that is, those events will perform the default processing. However, when you specify the action for a new event type using □WS, any actions previously defined for other event types will remain as they were.

Examples using Event Names

Ignore MouseDown (1) event (APL will perform the default processing for you)

```
'F1' □WS 'Event' 'MouseDown' 0
```


Terminate □DQ on MouseDown

```
'F1' □WS 'Event' 'MouseDown' 1
```

Invoke callback function F00 on MouseDown, the first element of the right argument to F00 will contain a *namespace reference* to F1

```
'F1' □WS 'Event' 'onMouseDown' 'F00'
```

Invoke callback function F00 on MouseDown, the first element of the right argument to F00 will contain the *character vector* 'F1'

```
'F1' □WS 'Event' 'MouseDown' 'F00'
```

Invoke callback function F00 on MouseDown and MouseUp

```
'F1' □WS 'Event' ('onMouseDown' 'onMouseUp') 'F00'
```

Invoke callback function F00 with ('THIS' 1) as its left-argument on MouseDown

```
'F1' □WS 'Event' 'onMouseDown' 'F00' ('THIS' 1)
```

Invoke callback function F00 with ('THIS' 1) as its left-argument on MouseDown, MouseUp and MouseMove

```
EV ← 'onMouseDown' 'onMouseUp' 'onMouseMove'
'F1' □WS 'Event' EV 'F00' ('THIS' 1)
```

Execute the expression COUNT++1 on MouseDown

```
'F1' □WS 'Event' 'MouseDown' '⍎COUNT++1'
```

Execute the expression COUNT++1 on MouseDown, MouseUp and MouseMove

```
EV ← 'MouseDown' 'MouseUp' 'MouseMove'
'F1' □WS 'Event' EV '⍎COUNT++1'
```

Examples using Event Numbers

Ignore MouseDown (1) event (APL will perform the default processing for you)

```
'F1' □WS 'Event' (1 0)
'F1' □WS 'Event' 1 0     A Ditto
```

Terminate □DQ on MouseDown

```
'F1' □WS 'Event' (1 1)
'F1' □WS 'Event' 1 1      A Ditto
```

Call function FOO on MouseDown

```
'F1' □WS 'Event' (1 'FOO')
'F1' □WS 'Event' 1 'FOO'      A Ditto
```

Call function FOO on MouseDown and MouseUp

```
'F1' □WS 'Event' ((1 2) 'FOO')
'F1' □WS 'Event' (1 2) 'FOO'      A Ditto
'F1' □WS 'Event' 1 2 'FOO'      A Ditto
'F1' □WS 'Event' (1 'FOO')(2 'FOO') A Ditto
```

Call function FOO with ('THIS' 1) as its left-argument on MouseDown

```
'F1' □WS 'Event' (1 'FOO' ('THIS' 1))
'F1' □WS 'Event' 1 'FOO' ('THIS' 1) A Ditto
```

Call function FOO with ('THIS' 1) as its left-argument on MouseDown and MouseUp

```
'F1' □WS 'Event' ((1 2) 'FOO' ('THIS' 1))
'F1' □WS 'Event' (1 2) 'FOO' ('THIS' 1) A Ditto
'F1' □WS 'Event' 1 2 'FOO' ('THIS' 1) A Ditto
'F1' □WS 'Event' 1 2 'FOO' ('THIS' 1) A Ditto
```

Execute the expression COUNT+←1 on MouseDown

```
'F1' □WS 'Event' 1 '⌞COUNT+←1'
```

Execute the expression COUNT+←1 on MouseDown, MouseUp and MouseMove

```
'F1' □WS 'Event' (1 2 3) '⌞COUNT+←1'
'F1' □WS 'Event' 1 2 3 '⌞COUNT+←1'      A Ditto
```

User defined Events

In addition to the standard events supported directly by Dyalog APL, you may specify your own events. For these, you **must** use event numbers; user-defined event names are not allowed.

You may use any numbers not already defined, but it is strongly recommended that you choose numbers greater than 1000 to avoid potential conflict with future releases of Dyalog APL.

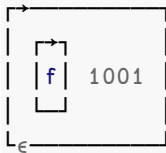
You can only **generate** user-defined events under program control with `□NQ`.

EXAMPLES

```

▼ foo m
[1]   □SE.UCMD'display m'
▼
    'f'□WC'Form' ('Event' 1001 'foo')
    f.Event
1001  #.foo
    □NQ 'f' 1001

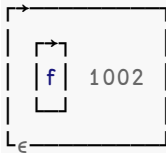
```



```

    'f' □WS 'Event' 1002 'foo'
    f.Event
1001  #.foo   1002  #.foo
    □NQ 'f' 1002

```



Notice that if you use the `onEvent` syntax, the event property reports the event type as you specified, but the callback function receives just the number as before.

```

f.on1003←'foo'
f.Event
1001  #.foo   1002  #.foo   on1003  #.foo
    □NQ 'f' 1003

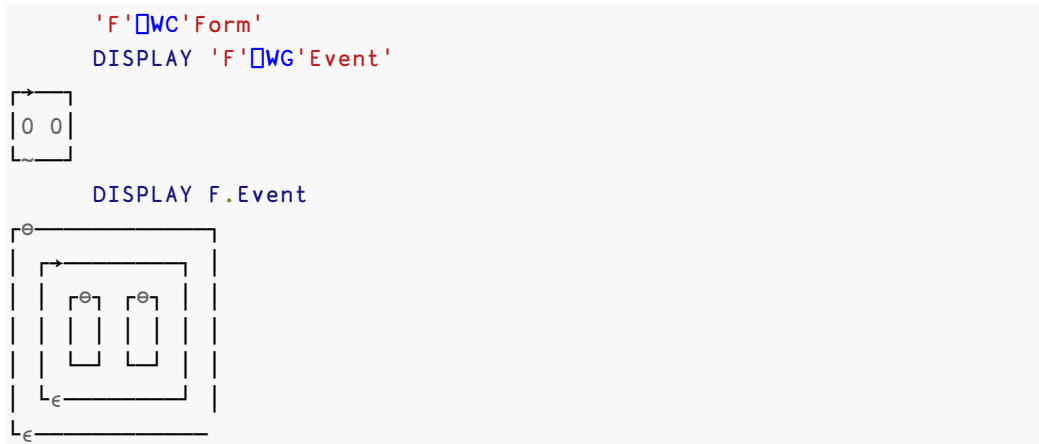
```



Notes

Resetting (clearing) the Event Property

If no events are set, the result obtained by `WC` and the result obtained by referencing Event directly are different:



To reset the Event property, the same (different) values must be used accordingly:

```
f.Event ← 0 p< ' ' ' '
```

or

```
'f'WS'Event' 0 0
```

onEvent Syntax with Event Numbers

If you use the *onEvent* syntax with *built-in* event numbers, the effect is the same as if you had used the event name. This does not apply to user-defined events.

EXAMPLE

```

    'f' □WC 'Form'
    f.on2 ← 'foo'
    f.Event
    onMouseUp #.foo
    ▼foo▼
    ▼ foo m
[1]   □SE.UCMD 'display m'
    ▼

```

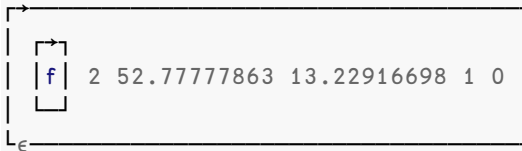


This differs from the behaviour when you use event number normally:

```

    'f' □WC 'Form'
    'f' □WS 'Event' 2 'foo'
    f.Event
    2 #.foo

```



Callback Names

When you query the Event property using □WG, names of callbacks associated with events are reported exactly as they were set. When you reference the Event property, the names are reported as absolute pathnames.

```

)NS x
#.x
)CS x
#.x
'f'␣WC'form'
f.onMouseUp←'foo'
f.Event

|onMouseUp|#.x.foo|

'f'␣WC'event'

|onMouseUp|foo|

)CS
#
#.x.f.Event

|onMouseUp|#.x.foo|

'#.x.f'␣WC'Event'

|onMouseUp|x.foo|

```

Spelling Event Names

When using regular event names, case is unimportant. For example, the system will accept 'MouseUp', 'MOUSEUP' or even 'mOuSeUp'. When using the *onEvent* syntax, case is critical. The 'on' must be in lower-case and the case of the event name must be spelled exactly as documented. In all cases, the event name will be reported using the documented spelling.

Special Case for All events

The event number 0 and the event name 'All', are convenient shorthands to associate a particular action (such as a callback) with **all** the built-in events supported by an object.

EXAMPLE

```
'f' WC 'Form' ( 'Event' 'All' 1)
f.Event
All 1
  DQ 'f'
f Create 1
```

6.197 EventList Property

Applies To

Description

This is a read-only property that reports the names of all the events supported by an object.

6.198 ExecuteJavaScript Method 839

Applies To: [Section 6.307](#)

Description

This method is used to execute JavaScript in an [Section 6.307](#) object.

The argument to ExecuteJavaScript is a single item as follows:

[1]	Code	character vector containing JavaScript code

The shy result of ExecuteJavaScript is currently 1; this may change.

EXAMPLE

```
hr.ExecuteJavaScript 'alert("Hello")'
```

6.199 ExitApp Event 132

Applies To: [Section 6.489](#)

Description

If enabled, this event is reported when the user attempts to terminate a Dyalog APL/W application from the Windows Task List.

The Windows Task list displays the names of all running applications. The name displayed for a Dyalog APL/W application is defined by the [Section 6.66](#) property of the system object `Root`. If you fail to define this property, there will be no entry for the application in the Task List.

If you wish to prevent the user from terminating your application from the Windows Task List, you may disable this event by setting its action code to `-1`. However, if you do this, your user may be puzzled as to why the operation does not work as expected. An alternative is to attach a callback function to the event which displays a message box. Not only does this allow you to provide user feedback, but you can provide confirm/cancel options. If your callback function returns a zero, your application will not be terminated.

Note that this event only provides for termination via the Windows Task List. See also the [Section 6.200](#) event.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ExitApp' or 132

6.200 ExitWindows Event 131

Applies To: [Section 6.489](#)

Description

If enabled, this event is reported when the user attempts to terminate the Windows Operating System. When this is done, Windows gives all running applications the opportunity to prevent it. Typically, an application that has unsaved changes will display a dialog box warning the user of this situation and offering the opportunity to cancel the termination. The default action for this event is to allow Windows to close. You can

prevent this by returning a zero from a callback function. You can also prevent the user from closing Windows down by disabling the event altogether. This is achieved by setting its action code to `-1`. In most cases this is less preferable than the callback method as it does not allow you to inform the user as to why Windows won't terminate.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ExitWindows' or 131
[3]	Flag	0 or 1

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

6.201 Expanding Event 302

Applies To: [Section 6.263](#), [Section 6.629](#)

Description

If enabled, this event is reported by a [Section 6.263](#) or a [Section 6.629](#) object just before it is about to expand to show additional rows or children of the current item.

In a Grid, this occurs when the user clicks the picture or tree line in the row title.

In a TreeView, this occurs when the user double-clicks the item label or clicks in the button or on the tree line to the left of the item label.

The default processing for the event is to expand the tree at the corresponding point.

You may disable the expand operation by setting the action code for the event to `-1`. You may also prevent the expand from occurring by returning 0 from a callback function. You may expand an object dynamically under program control by calling `Expanding` as a method.

The event message reported as the result of `⌈DQ`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Expanding' or 302
[3]	Item number	Integer. The index of the row or item.

6.202 ExportedFns Property

Applies To: [Section 6.420](#)

Description

This property specifies the functions to be exposed as methods by an [Section 6.420](#) object.

ExportedFns may be set to 0 (none), 1 (all), or a vector of character vectors containing the names of the functions to be exported.

There are certain important restrictions concerning the type of function that you can export as a method.

Firstly, only top-level defined functions within the [Section 6.420](#) may be exported; you cannot export functions in other namespaces including sub-namespaces.

Furthermore, you may not export defined operators, dfns, external functions, or functions created by function assignment.

Finally, OLE does not support the concept of a dyadic function, so your exported functions must be niladic, monadic, or take an optional left argument; they may not be explicitly dyadic.

If you wish to export a new function from your [Section 6.420](#), and ExportedFns is not 1, you must explicitly reset the value of the ExportedFns property before you re-save the workspace.

6.203 ExportedVars Property

Applies To: [Section 6.420](#)

Description

This property specifies the variables to be exposed as properties by an [Section 6.420](#) object.

ExportedVars may be set to 0 (none), 1 (all), or a vector of character vectors containing the names of the variables to be exported.

Note that you may not export external variables or shared variables, or variables in other namespaces.

If you wish to export a new variable from your [Section 6.420](#), and ExportedVars is not 1, you must explicitly reset the value of the ExportedVars property before you re-save the workspace.

6.204 Expose Event 32

Applies To

Description

If enabled, this event is reported when part or all of the object's window is exposed to view. Under normal circumstances, APL repaints the exposed region automatically. However, if you have drawn unnamed graphical objects (which are **not** managed by APL) you should use this event to redraw them when required. Note that APL will itself repaint any **named** objects in the region before reporting the event.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Expose' or 32
[3]	Y	y-position of top-left corner of exposed region
[4]	X	x-position of top-left corner of exposed region
[5]	H	height of exposed region
[6]	W	width of exposed region

This event **cannot** be disabled by setting its action code to $\bar{1}$. Similarly, setting the result of a callback function to 0 has no effect on it.

6.205 FCol Property

Applies To

Description

This property defines the foreground colour(s) of an object.

For non-graphical objects, FCol specifies a single colour. For graphical objects with more than one constituent part, it may specify a set of background colours, one for each part. A single colour is represented by a single number which refers to a standard colour, or by a 3-element vector which defines a colour explicitly in terms of its red, green and blue intensities.

If FCol is 0 (which is the default) the foreground colour is defined by your current colour scheme for the object in question. For example, if you select red as your MS-Windows "Button Text" colour, you will by default get red writing on all your [Section 6.55](#) objects, simply by not specifying FCol or by setting it to 0.

A negative value of FCol refers to a standard MS-Windows colour as described below. Positive values are reserved for a possible future extension.

FCol	Colour Element	FCol	Colour Element
0	Default	-11	Active Border
-1	Scroll Bars	-12	Inactive Border
-2	Desktop	-13	Application Workspace
-3	Active Title Bar	-14	Highlight
-4	Inactive Title Bar	-15	Highlighted Text
-5	Menu Bar	-16	Button Face
-6	Window Background	-17	Button Shadow
-7	Window Frame	-18	Disabled Text
-8	Menu Text	-19	Button Text
-9	Window Text	-20	Inactive Title Bar Text
-10	Active Title Bar Text	-21	Button Highlight

If instead, FCol contains a 3-element vector, it specifies the intensity of the red, green and blue components of the colour as values in the range 0-255. For example, (255 0 0) is red and (255 255 0) is yellow.

Note that if the colour specified by FCol would normally be rendered as a dithered colour, it is instead converted to the nearest pure colour available on the device. The

actual colour realised also depends upon the capabilities of the display adapter and driver, and the current Windows colour map.

For a [Section 6.55](#), [Section 6.121](#), [Section 6.185](#), [Section 6.341](#), [Section 6.346](#), [Section 6.382](#) and [Section 6.384](#), FCol refers to the colour of the **text** in the object. [Section 6.49s](#) around these objects (where applicable) are drawn using the standard Windows colour. For a [Section 6.573](#) object however, FCol specifies the colour of its border.

For the [Section 6.190](#), [Section 6.450](#) and [Section 6.472](#) objects, FCol specifies the colour of the line drawn around the perimeter of the object. If a dashed or dotted line is used ([Section 6.360](#) 1-4) the "gaps" in the line are drawn using the colour specified by [Section 6.45](#), or are left undrawn if [Section 6.45](#) is not specified. For the [Section 6.365](#) object, FCol specifies the colour in which the markers are drawn. For the graphics objects [Section 6.190](#), [Section 6.450](#), [Section 6.472](#) and [Section 6.605](#), FCol may be a vector of 3-element vectors specifying a set of colours for the constituent parts of the object. For example, a [Section 6.450](#) object consisting of four polygons, may have a FCol property of four 3-element vectors. In addition, for graphics objects, FCol is used in place of [Section 6.214](#) if the latter is not specified.

6.206 FieldType Property

Applies To: [Section 6.56](#), [Section 6.143](#), [Section 6.185](#), [Section 6.341](#), [Section 6.563](#)

Description

The FieldType property controls data conversion, formatting and validation.

For [Section 6.185](#), [Section 6.341](#) and [Section 6.563](#) objects, FieldType controls how the [Section 6.641](#) property of these objects is interpreted.

FieldType is a character vector. If it is empty (the default), the [Section 6.605](#) or [Section 6.341](#) object is a standard text object with no special formatting or, in the case of an [Section 6.185](#), no input validation. For a [Section 6.143](#), an empty FieldType implies the default which is 'Date'.

For a [Section 6.143](#), FieldType may be one of the following:

'Date '	Uses Windows "short date" format
'DateCentury '	Uses Windows "short date" format but with a 4-digit year regardless of user preference
'LongDate '	Uses Windows "long date" format
'Time '	Uses Windows time format
'Custom '	Uses a special format defined by the CustomFormat property

The value of the date or time is represented by the [Section 6.141](#) property. Note that all validation is performed by the object itself, and it is impossible to enter an invalid value.

For an [Section 6.185](#), [Section 6.341](#) and [Section 6.563](#), if FieldType is defined, the contents of the object are defined by its [Section 6.641](#) property, which is a number, rather than by its [Section 6.606](#) property, and special formatting and validation rules are applied. FieldType may be one of the following :

'Numeric '	Simple numeric formatting and validation
'LongNumeric '	Uses Windows number format
'Date '	Uses Windows "short date" format
'LongDate '	Uses Windows "long date" format
'Currency '	Uses Windows currency format
'Time '	Uses Windows time format
'Char '	No validation, forces Value to be a character vector

FieldType 'Char ' only affects an [Section 6.185](#) object. When the user enters data into a standard single-line [Section 6.185](#) object, the [Section 6.641](#) property is set to a number if the contents are numeric, or to a character vector if the contents do not represent a valid number. If FieldType is 'Char ', the [Section 6.641](#) property is always set to a character vector, regardless of the type of the field contents.

If FieldType is 'Numeric ', the object displays the number defined by its [Section 6.641](#) property rounded to the number of decimal places specified by its [Section 6.147](#) property. The decimal separator character used will be as specified by the Number format in the user's International Control Panel settings. If the object is an [Section 6.185](#) object, the user is prevented from entering anything but a valid number. The number of decimal digits is also restricted to [Section 6.147](#). When the user "leaves" the object, the number is re-formatted.

If `FieldType` is `'LongNumeric'`, the object displays the number specified by its [Section 6.641](#) property according to the Number format in the user's International Control Panel settings. This format specifies the 1000 separator, decimal separator, decimal digits and whether or not a leading zero is inserted. If the object is an [Section 6.185](#) object, the user is prevented from entering anything but a valid number. However, the character specified for the 1000 separator is ignored and may be entered anywhere in the number. When the user "leaves" the object, the number is re-formatted correctly.

If the `FieldType` is `'Currency'`, the object displays the number specified by its [Section 6.641](#) property according to the Currency format in the user's International Control Panel settings. This specifies the currency symbol and placement, the way in which a negative value is displayed, and the number of decimal places. If the object is an [Section 6.185](#) object, the user is restricted to entering a "reasonable" value. When the user leaves the object, the number is reformatted correctly.

If the `FieldType` is `'Date'`, the [Section 6.641](#) property represents the number of days since January 1st 1900 and is displayed using the "short date" format specified by the user's International Control Panel settings. If the object is an [Section 6.185](#) object, the user is restricted to entering a "reasonable" date. The object will accept any numeric triplet separated by slash(/), colon(:) or space characters but checks that the day number and month number lie in the range 1-31 and 1-12 respectively and will not allow the user to enter a digit that would invalidate this. (Note that the position within the triplet of the day, month and year are as specified by the Windows short date format). However, the user is not prevented from entering an invalid date such as 31st September.

If the `FieldType` is `'LongDate'`, the [Section 6.641](#) property represents the number of days since January 1st 1900 and is displayed using the "long date" format specified by the user's International Control Panel settings. If the object is an [Section 6.185](#) object, its appearance and behaviour automatically switches to `FieldType 'Date'` when it has the input focus and back again when it loses the focus. This allows the user to edit or input a date in a more convenient form.

If the `FieldType` is `'Time'`, the [Section 6.641](#) property represents the number of seconds since midnight and is displayed using the time format specified by the user's International Control Panel settings.

When the user attempts to move the input focus away from the object, the contents are validated. If they cannot be converted to a valid number, date, or time, the object generates a [Section 6.38](#) event, or, if the object is associated with a [Section 6.263](#), the [Section 6.263](#) (and not the [Section 6.185](#) object) generates a [Section 6.74](#) event. See the descriptions of these events for further details.

Note that for [Section 6.185](#), [Section 6.341](#) and [Section 6.563](#) objects, `FieldType` may only be specified when you create an object using `WC`.

6.207 File Property

Applies To

Description

Specifies the name of a file associated with an object.

For an [Section 6.21](#), [Section 6.47](#), [Section 6.136](#) or [Section 6.309](#) object, this property is either a simple character vector or a 2-element nested vector.

If it is simple, `File` specifies the name of the associated bitmap (.BMP), icon (.ICO) or cursor (.CUR) file.

If it is nested, the first element specifies the name of a DLL or ([Section 6.309](#) only) EXE and the second element identifies the particular bitmap, icon or cursor in that file. The identifier may be its *name* (a character string), its *resource id* (a non-zero positive integer) or ([Section 6.309](#) only), its *index* (0 or negative integer) within the file. As a special case, if the name of the file is an empty vector, the object is loaded from DYALOG.EXE or the Dyalog resource DLL. The name of the latter varies according to the version installed but begins DYARES. In this case, the identifier must be a name or resource id; indexes are not supported.

For a [Section 6.385](#) object, `File` must be simple and specifies the name of a metafile (.WMF) file. For a [Section 6.488](#) object, `File` must be simple and specifies the name of a Rich text Format (.RTF) file.

When applied to a [Section 6.208](#) object, `File` contains the name (or names) of the selected file (or files) depending upon the value of its [Section 6.577](#) ('Single' or 'Multi').

6.208 FileBox Object

Purpose: Prompts user to select a file.

Parents ActiveXControl CoolBand Form Grid OLEServer PropertyPage PropertySheet Root StatusBar SubForm TCPSocket ToolBar ToolControl

Children Timer

Properties Type Caption Directory Filters File FileMode Style Event Index Data
EdgeStyle KeepOnClose MethodList ChildList EventList PropList

Methods Detach Wait

Events Close Create FileBoxOK FileBoxCancel Select

Description

The FileBox object implements the standard Windows FileSelection Dialog Box. This is a "modal" object. When you create a FileBox with *Dyalog APL Language: Wc*, it is initially invisible and the user cannot interact with it. To use it, you must execute *Dyalog APL Language: Dq* with the name of the FileBox as its right argument. This causes the FileBox to be displayed. During the "local" *Dyalog APL Language: Dq* the user may interact **only** with the FileBox, or with other applications. When the user terminates the operation (by pressing the "Save", "Open", or "Cancel" Buttons, or by closing the window) the "local" *Dyalog APL Language: Dq* terminates, and the FileBox disappears.

When the "local" *Dyalog APL Language: Dq* is terminated, the FileBox generates either an [Section 6.210](#)(71) or [Section 6.209](#)(72) event. The former is generated when the user presses the "Save" or "Open" button; the latter when the user presses the "Cancel" button or closes the FileBox.

The [Section 6.211](#) property is a character vector which indicates the mode in which the selected file is going to be opened. [Section 6.211](#) may be 'Read' (the default) or 'Write'. If [Section 6.211](#) is 'Write', files listed in the File Selection Box are "greyed", although they may still be selected.

The [Section 6.66](#) property determines the text that appears in the title bar of the FileBox window. If undefined, [Section 6.66](#) defaults to "Save As" if [Section 6.211](#) is 'Write' or to "Open" if [Section 6.211](#) is 'Read'. Similarly, if [Section 6.211](#) is 'Write', the button captions are "Save" and "Cancel", and if [Section 6.211](#) is 'Read' the button caption are "Open" and "Cancel".

The [Section 6.159](#) property contains a simple character vector which specifies the initial directory from which a list of suitable files is displayed.

```
'F' □WC 'FileBox' 'The FileBox Object' 'C:\WDYALOG'
□DQ 'F'
```

The [Section 6.577](#) property specifies whether the user may choose a single file name ('Single' which is the default) or several file names ('Multi').

The [Section 6.215](#) property is a nested scalar or vector containing a list of filters. Each filter is a 2-element vector of character vectors which contain a file type mask and a file type description respectively. The file type descriptions appear in a drop-down combo box labelled "List File of Type". When the user selects one of these, the currently selected directory is searched for files which match the corresponding mask. The default value of [Section 6.215](#) is an empty vector. This gives a file type mask of "." and a file type description of "All Files (.)". Hence an empty vector is equivalent to `(c'*. *' 'All Files (*.*)')`.

The [Section 6.319](#) property determines which of the filters is initially selected. Its default value is `10`.

Note that when *Dyalog APL Language: Dq* terminates with [Section 6.210](#), the [Section 6.207](#), [Section 6.159](#), and [Section 6.319](#) properties are updated to reflect the contents of the fields within the FileBox.

The operating system imposes limits on both the length of the name of the file, and on the total path length. In version 20.0 attempting to set the File or Directory Properties to too long a name will generate a DOMAIN ERROR, while attempting to use too long a File name within the FileBox will result in the appearance of an error MessageBox.

6.209 FileBoxCancel Event 72

Applies To: [Section 6.51](#), [Section 6.208](#)

Description

If enabled, this event is reported when a [Section 6.208](#) is closed because the user has pressed the "Cancel" button or closed it.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'FileBoxCancel' or 72
[3]	File name	character vector containing the name of the currently selected file (empty if none)

6.210 FileBoxOK Event 71

Applies To: [Section 6.51](#), [Section 6.208](#)

Description

If enabled, this event is reported when a [Section 6.208](#) is closed because the user has pressed the "Open" or "Save" button.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'FileBoxOK' or 71
[3]	File name	character vector containing the name of the currently selected file (empty if none)

6.211 FileMode Property

Applies To: [Section 6.208](#)

Description

The FileMode property applies only to a [Section 6.208](#) object. It indicates the mode in which the selected file is going to be opened. It is a character vector containing 'Read' (the default) or 'Write'. If FileMode is 'Write', files listed in the File Selection Box are greyed, although they may still be selected.

6.212 FileRead Method 90

Applies To: [Section 6.47](#), [Section 6.136](#), [Section 6.309](#), [Section 6.385](#), [Section 6.488](#)

Description

This method causes the object to be recreated from the file named in its [Section 6.207](#) property.

The FileRead method is niladic.

[1]	Object	ref or character vector
[2]	Event	'FileRead' or 90

If you attach a callback function to this event and have it return a value of 0, the object will not be recreated from file.

6.213 FileWrite Method 91

Applies To: [Section 6.47](#), [Section 6.136](#), [Section 6.309](#), [Section 6.385](#), [Section 6.488](#)

Description

This method causes the object to be written to the file named in its [Section 6.207](#) property. For the Bitmap and Icon objects this method will write a file of type .BMP and .ICO to a file with the appropriate extension. If [Section 6.207](#) specifies any other extension, the method will fail with a DOMAIN ERROR:

DOMAIN ERROR: This object cannot be saved to this type of file.

The FileWrite method is niladic.

If you attach a callback function to this event and have it return a value of 0, the object will not be written to file. You could use this to avoid overwriting an existing file.

6.214 FillCol Property

Applies To: [Section 6.94](#), [Section 6.190](#), [Section 6.450](#), [Section 6.472](#)

Description

This property defines the fill colour in a graphics object.

If [Section 6.231](#) is 0 (solid fill) FillCol defines the colour with which the object is filled. If [Section 6.231](#) is in the range 1-6 (pattern fill) it defines the colour of the lines that make up the pattern. The areas between the lines are filled using the colour specified by [Section 6.45](#), or are left undrawn (transparent) if [Section 6.45](#) is not specified. If [Section 6.231](#) contains the name of a [Section 6.47](#) object, the value of FillCol is ignored.

A single colour is represented by a single number which refers to a standard colour, or by a 3-element vector which defines a colour explicitly in terms of its red, green and blue intensities. A negative value of FillCol refers to a standard MS-Windows colour as described below. Positive values are reserved for a possible future extension.

FillCol	Colour Element	FillCol	Colour Element
0	Default	-11	Active Border
-1	Scroll Bars	-12	Inactive Border
-2	Desktop	-13	Application Workspace
-3	Active Title Bar	-14	Highlight
-4	Inactive Title Bar	-15	Highlighted Text
-5	Menu Bar	-16	Button Face
-6	Window Background	-17	Button Shadow
-7	Window Frame	-18	Disabled Text
-8	Menu Text	-19	Button Text
-9	Window Text	-20	Inactive Title Bar Text
-10	Active Title Bar Text	-21	Button Highlight

If instead, FillCol contains a 3-element vector, it specifies the intensity of the red, green and blue components of the colour as values in the range 0-255. For example, (255 0 0) is red and (255 255 0) is yellow. Note that the colour realised depends upon the capabilities of the display adapter and driver, and the current Windows colour map.

FillCol may also be a vector of 3-element vectors specifying a set of colours for the constituent parts of the object. For example, a [Section 6.450](#) object consisting of four polygons, may have a FillCol property of four 3-element vectors.

6.215 Filters Property

Applies To: [Section 6.208](#)

Description

The Filters property is a nested scalar or vector containing a list of filters. Each filter is a 2-element vector of character vectors which contain a file type mask and a file type description respectively. The file type descriptions appear in a drop-down combo box labelled "List Files of Type". When the user selects one of these, the currently selected directory is searched for files which match the corresponding mask. The default value of Filters is an empty vector. This gives a file type mask of "." and a file type description of "All Files (.)". Hence an empty vector is equivalent to (c'*. *' 'All Files (*.*)').

6.216 FireOnce Property

Applies To: [Section 6.613](#)

Description

This property specifies one-off behaviour for a [Section 6.613](#) object. It has the value 0, 1 or 2.

Setting FireOnce to 1, will cause the [Section 6.613](#) to generate a single event and no more unless it is reset. After generating the single event, FireOnce is automatically set to 2 and this change occurs prior to the invocation of a callback function.

Setting FireOnce to 2 will cause the [Section 6.613](#) to behave as if it has already raised its single event.

FireOnce honours the value of the [Section 6.5](#) property but does not change it. So setting FireOnce to 1 when [Section 6.5](#) is 0 will not immediately cause an event. If [Section 6.5](#) is subsequently set to 1, the single [Section 6.614](#) event will then occur.

6.217 FirstDay Property

Applies To: [Section 6.58](#)

Description

The FirstDay property specifies the day that is considered to be the first day of the week and which appears first in the [Section 6.58](#).

FirstDay is an integer whose value is in the range 0-6. The default value for FirstDay depends upon your International Settings, but in most countries is 0 meaning Monday.

6.218 Fixed Property

Applies To: [Section 6.222](#)

Description

This property specifies whether or not a font represented by a [Section 6.222](#) object is fixed-width or proportional. It is either 0 (fixed-width) or 1 (proportional). There is no default; the value of this property reflects the characteristic of the selected font.

6.219 FixedOrder Property

Applies To: [Section 6.127](#)

Description

The FixedOrder property specifies whether or not the [Section 6.127](#) displays [Section 6.126](#) in the same order.

FixedOrder is a single number with the value 0 (user may re-order bands) or 1 (user may not re-order bands); the default is 0.

If FixedOrder is 1, the user may move bands to different rows, but the band order is static.

6.220 FlatSeparators Property

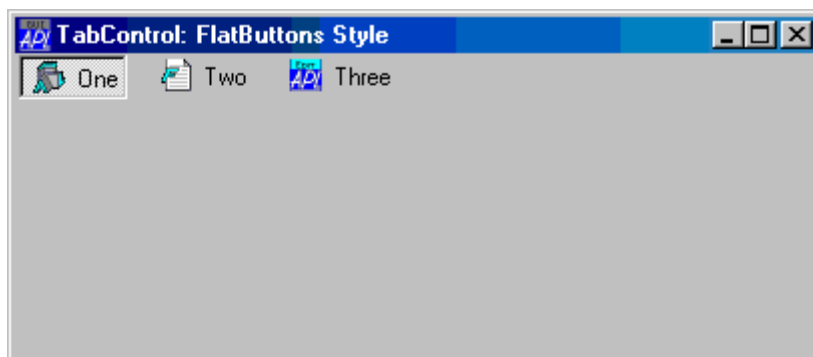
Applies To: [Section 6.585](#)

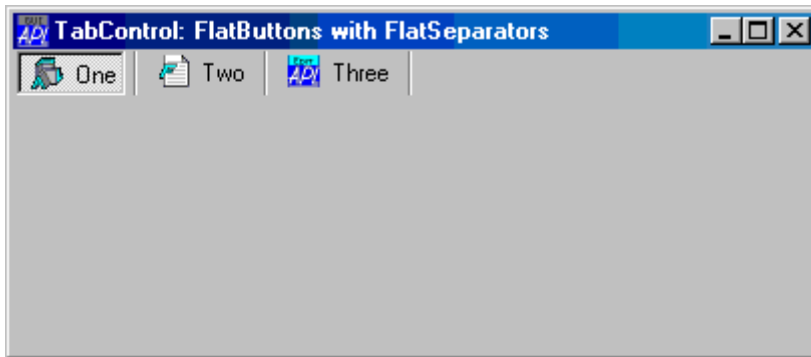
Description

The FlatSeparators property specifies whether or not separators are drawn between buttons in a [Section 6.585](#) object. FlatSeparators only affects a [Section 6.585](#) if [Section 6.577](#) is 'FlatButtons' and is otherwise ignored.

FlatSeparators is a single number with the value 0 (no separators) or 1 (separators); the default is 0.

The pictures below illustrate the effect of FlatSeparators on the appearance of a [Section 6.585](#) object.





6.221 Flush Method 135

Applies To: [Section 6.489](#)

Description

This method forces any objects that have been created but not yet shown to be displayed. Normally, Dyalog APL/W buffers the display of new objects unless they are being created by a callback function. This event can be used to override the buffering.

The Flush method is niladic.

6.222 Font Object

Purpose: Loads a font resource

Parents ActiveXControl Animation Bitmap Button ButtonEdit Calendar Combo ComboEx CoolBand DateTimePicker Edit Form Grid Group Label List ListView MDIClient Metafile OLEServer Printer ProgressBar PropertyPage PropertySheet RichEdit Root Scroll Spinner Static StatusBar SubForm TabBar TCPSocket TipField ToolBar ToolControl TrackBar TreeView UpDown

Children Timer

Properties Type PName Size Fixed Italic Underline Weight Rotate CharSet Coord Data Handle KeepOnClose MethodList ChildList EventList PropList

Methods Detach ChooseFont

Events Close Create FontOK FontCancel Select

Description

This object loads a Windows font into memory ready for use by another object.

The characteristics of the font are specified by its properties as follows :

<u><i>PName</i></u>	A character vector containing the name of the font face. The default is 'System' . Note that case is ignored when you specify the name, although it will be returned correctly by <i>Dyalog APL Language: Wg</i> .
<u><i>Size</i></u>	An integer that specifies the character height of the font in pixels.
<u><i>Fixed</i></u>	A Boolean value that specifies whether the font is fixed-width (1) or proportional (0).
<u><i>Italic</i></u>	A Boolean value that specifies whether the font is italicised (1) or not (0).
<u><i>Underline</i></u>	A Boolean value that specifies whether the font is underlined (1) or not (0).
<u><i>Weight</i></u>	An integer in the range 0-1000 that specifies how bold or heavy the font is (1000 = most bold).
<u><i>Rotate</i></u>	A numeric scalar that specifies the angle of rotation of the font in radians. The angle is measure from the x-axis in a counter-clockwise direction.
<u><i>CharSet</i></u>	An integer that specifies the character encoding.

The [Section 6.128](#) property may be set to 'Pixel', 'ScaledPixel' or 'RealPixel' when the object is created, but [Section 6.128](#) may not subsequently be changed.

If Coord is 'Pixel', it is interpreted as either 'RealPixel' or 'ScaledPixel' according to the value of the **Dyalog_Pixel_Type** parameter, which is either ScaledPixel or RealPixel. See *Installation/Configuration: Dyalog Pixel Type*.

If this parameter is not specified, the default is RealPixel. So by default, when you set Coord to Pixel, it will be treated as RealPixel.

If you are using 'ScaledPixel', this means that your fonts will also be scaled up automatically, as well as the sizes of the controls in which they are used.

When you ask Windows to allocate a font, you may specify as many or as few of these properties as you wish. Windows actually supplies the font that most closely matches the attributes you have specified. The matching rules it uses are complex, and may be found in the appropriate Windows documentation.

The values of the above properties after *Dyalog APL Language: Wc* or *Dyalog APL Language: Ws* reflect the attributes of the font which has been allocated by Windows, and not necessarily the values you have specified. Furthermore, it is possible that changing the value of one property will cause the values of others to be changed.

6.223 FontCancel Event 242

Applies To

Description

If enabled, this event is reported when the user has pressed the *Cancel* button or closed the font selection dialog box that is displayed by the ChooseFont method.

The event message reported as the result of □□□, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'FontCancel ' or 242

6.224 FontList Property

Applies To: [Section 6.455](#), [Section 6.489](#)

Description

The FontList property is a read-only property (you cannot set its value) that provides a list of available fonts.

Its value is a vector (1 per font) of 8-element character vectors, each of which is as follows :

[1]	Face name (character vector)
[2]	Character height in "points" (integer)
[3]	Fixed width or not (Boolean)
[4]	Italic or not (Boolean)
[5]	Underline or not (Boolean)
[6]	Weight (integer)
[7]	Angle of rotation (integer)
[8]	Character set (see CharSet)

EXAMPLE

```

↑'.\DWG'FontList'
System          16 0 0 0 700 0 0
Terminal        12 1 0 0 400 0 255
Fixedsys        15 1 0 0 400 0 0
Roman          37 0 0 0 400 0 255
Script         36 0 0 0 400 0 255
Modern          37 0 0 0 400 0 255
Small Fonts     3 0 0 0 400 0 0
Courier        13 1 0 0 400 0 0
Serif          13 0 0 0 400 0 0
Dyalog Alt     16 1 0 0 400 0 0
Dyalog Std     16 1 0 0 400 0 0

```

Note that the list of fonts obtained from FontList for a [Section 6.455](#) object will include TrueType fonts and printer fonts but will exclude screen fonts. FontList for [Section 6.489](#) will include TrueType fonts and screen fonts, but exclude printer-only fonts. The two lists will therefore (typically) be different.

6.225 FontObj Property

Applies To

Description

The FontObj property associates a font with an object. It specifies either the name of or a ref to a [Section 6.222](#) object

If unspecified, the default value for FontObj is an empty character vector. For most objects, this setting implies that the font used in the object is **inherited** from its parent object. However, [Section 6.127](#), [Section 6.382](#), [Section 6.383](#), [Section 6.574](#), [Section 6.616](#), [Section 6.621](#), and [Section 6.624](#) objects do not inherit their font.

Note that the default value of FontObj for [Section 6.489](#) is also an empty character vector and that this implies the Windows default GUI font, which is a Windows user preference setting.

Note however that it is not currently possible to specify the font for [Section 6.382](#) and [Section 6.384](#) objects which are the direct descendants of a [Section 6.383](#). Nor is it possible to specify the font used for the [Section 6.66](#) in a [Section 6.227](#).

6.226 FontOK Event 241

Applies To

Description

If enabled, this event is reported when the user has pressed the OK button in the font selection dialog box that is displayed by the ChooseFont method.

The event message reported as the result of `FontOK`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'FontOK' or 241
[3]	Font	nested vector
[4]	Colour	RGB triplet

The font specification in the 3rd element of the event message is a 7-element nested vector that describes the chosen font. See Font Object for further details.

The colour specification in the 4th element of the event message is a 3-element integer vector of RGB values for the colour chosen by the user.

6.227 Form Object

Purpose: This is a top-level window used to contain other objects (controls).

Parents ActiveXControl Form OLEClient OLEServer Root SubForm TCPSocket

Children ActiveXControl Animation Bitmap BrowseBox Button ButtonEdit Calendar Circle Clipboard ColorButton Combo ComboEx CoolBar Cursor DateTimePicker Edit Ellipse FileBox Font Form Grid Group HTMLRenderer Icon Image ImageList Label List ListView Locator Marker MDIClient Menu MenuBar Metafile MsgBox OCXClass OLEClient OLEServer Poly Printer ProgressBar PropertySheet Rect RichEdit Scroll SM Spinner Splitter Static StatusBar SubForm SysTrayItem TabBar TabControl TCPSocket Text Timer TipField Toolbar ToolControl TrackBar TreeView UpDown

Properties Type Caption Posn Size Coord State Border Active Visible Event Thumb Range Step VScroll HScroll Sizeable Moveable SysMenu MaxButton MinButton HelpButton OKButton SIPMode SIPResize FontObj BCol Picture OnTop IconObj CursorObj AutoConf YRange XRange Data TextSize EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles KeepOnClose Dockable Docked DockShowCaption DockChildren UndocksToRoot MaskCol AlphaBlend Redraw TabIndex PageSize MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ChooseFont Wait

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDoubleClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap StateChange DockStart DockMove DockRequest DockAccept DockEnd DockCancel Select FrameContextMenu DyalogCustomMessage1 FontOK FontCancel VScroll HScroll VThumbDrag HThumbDrag

Description

The [Section 6.452](#) property specifies the location of the **internal** top-left corner of the window relative to the top-left corner of the screen. If the window has a title bar and/or border, you must allow sufficient space for them. Similarly, the [Section 6.556](#) property specifies the internal size of the window excluding the title bar and border. The default for [Section 6.556](#) is 50% of the screen height and width. The default for [Section 6.452](#) places the Form in the middle of the screen.

Normally, a Form has a title bar, a system menu box, a border and maximise and minimise buttons. To disable the System Menu box, set [Section 6.580](#) to 0. To disable one or both of the maximise/minimise buttons, set [Section 6.369](#) and/or [Section 6.388](#) to 0.

The [Section 6.295](#) property specifies that a Question (?) button appears in the title bar of the Form. However, this does not apply if the Form has a maximise or minimise button which both take precedence. The user may obtain help by clicking on the

Question (?) button and then on a control in the Form. It is up to you to provide the help by responding to the [Section 6.294](#) event on the control.

By default, a Form may be moved and resized using the mouse. These actions are achieved by dragging on the title bar and border respectively. It follows that a Form that is [Section 6.398](#) **must** have a title bar, and one that is [Section 6.557](#) **must** have a border, regardless of the value of other properties. Also, if you specify any of [Section 6.580](#), [Section 6.369](#) or [Section 6.388](#), the window must have a title bar in which to place these controls. A title bar itself requires a border. To obtain a window without a title bar, you must therefore set [Section 6.398](#), [Section 6.580](#), [Section 6.369](#) and [Section 6.388](#) to 0. Note that setting [Section 6.66](#) does **not** force a title bar on the window.

If [Section 6.557](#) is 1, the window will have a double-line border, regardless of the values of other properties. If [Section 6.557](#) is 0, and any one or more of [Section 6.398](#), [Section 6.580](#), [Section 6.369](#), [Section 6.388](#) or [Section 6.49](#) is 1, the window will have a 1-pixel border. Only if all these properties are 0 will the window be borderless. To obtain a dialog box that may only be moved or closed, set [Section 6.49](#) to 2.

Note that the default value for [Section 6.66](#) is an empty character vector which results in a blank title.

To obtain a standard dialog box with 3-dimensional appearance, create a Form with [Section 6.49](#) set to 2 and [Section 6.184](#) set to 'Dialog', for example:

```
'F' [WC 'Form' 'Dialog Box' ('EdgeStyle' 'Dialog') ('Border' 2)]
```

The [Section 6.571](#) property has the value 0 if the window is currently displayed in its "normal" state, 1 if it is currently displayed as an icon, and 2 if it is currently maximised and displayed full-screen. This property does not just report the current state, but can be used to set the state under program control.

The [Section 6.646](#) and [Section 6.303](#) properties determine whether or not a Form has a vertical and horizontal scrollbar respectively. These properties are set to -1 to obtain a scrollbar. Their default value is 0 (no scrollbar). [Section 6.646](#) and [Section 6.303](#) may only be set when the object is created and may not subsequently be changed. The [Section 6.469](#) property is a 2-element vector that specifies the maximum value for the vertical and horizontal scrollbars respectively. The [Section 6.438](#) property is a 2-element vector that specifies the sizes of the thumbs in each scrollbar. The [Section 6.576](#) property is a 4-element vector that specifies the sizes of the small and large change. Its first two elements refer to the vertical scrollbar, elements 3 and 4 refer to the horizontal scrollbar. The [Section 6.608](#) property is a 2-element vector that both reports and sets the position of the thumb in the vertical and horizontal scrollbars

respectively. When the user attempts to move the thumb in one of the scrollbars, the Form generates a [Section 6.647](#) or [Section 6.304](#) event.

[Section 6.647](#) and [Section 6.304](#) cannot be changed using *Dyalog APL Language: Ws*. However, you can make a scrollbar disappear by setting the corresponding element of [Section 6.469](#) to 1, thus allowing you to dynamically switch the scrollbar off and on. Note however that doing so will change the size of the Form.

Setting the [Section 6.225](#) property on a Form does not affect the text in its title bar. However, the value of [Section 6.225](#) will (unless over-ridden) be inherited by all of the objects within the Form.

The background of the Form may be coloured using [Section 6.45](#). The default value for [Section 6.45](#) is the Windows Button Face colour unless [Section 6.184](#) is set to 'None' or 'Default' in which case it is the Windows background colour. Alternatively, the background of a Form can be defined using a [Section 6.47](#) or [Section 6.385](#) object whose name is defined by the [Section 6.447](#) property. A [Section 6.385](#) is automatically scaled to fit the Form. A [Section 6.47](#) can be tiled or scaled. See [Section 6.447](#) property for details.

The [Section 6.423](#) property is either 0 or 1. If it is 0, the Form assumes its normal position within the stack of windows on the screen and is only brought to the front when it receives the input focus. If [Section 6.423](#) is set to 1, the Form is always brought to the front even when it doesn't have the focus. If more than one Form has [Section 6.423](#) set to 1, the stacking order of this set of Forms is defined by the order in which they were created.

A Form can be created as a child of another Form. If so, it has the following characteristics:

- A child Form always appears on top of its parent Form (although it is not constrained by it)
- When you minimize a parent Form, its child Forms disappear.
- Making the parent Form invisible or inactive has no effect on a Child Form.

Note that the [Section 6.452](#) and [Section 6.556](#) properties of a child Form are expressed in screen coordinates and are not given relative to its parent.

6.228 Formats Property

Applies To: [Section 6.99](#)

Description

This is a "read-only" property that identifies the formats in which data is currently available in the clipboard. It is a vector of character vectors containing the names of the corresponding [Section 6.99](#) properties for which data may be obtained using *Dyalog APL Language: Wg*. In the following example data was copied to the Windows clipboard from Microsoft Excel. This product stores data in CF_Text and the older device-dependent CF_Bitmap formats. The latter excludes colour map information, so [Section 6.104](#) is not available.

```
'CL' □WC 'Clipboard'
'CL' □WG 'Formats'
Bits CMap Text
```

6.229 FormatString Property

Applies To: [Section 6.56](#), [Section 6.185](#), [Section 6.263](#), [Section 6.341](#), [Section 6.563](#)

Description

The FormatString property specifies one or more □FMT format specifications to be used to format data in an Grid or single-line Edit. When applied to a Grid, it is either a simple character vector that specifies the format specification for the entire Grid, or a vector of character vectors. If it is a vector, its elements are mapped to individual cells via the CellTypes property. When applied to an Edit object, FormatString must be a simple character vector.

APL derives the text to be displayed in a cell by calling □FMT with a left argument of the corresponding element of FormatString and a right argument of the cell value. If the format specification is invalid, the text displayed is blank.

When a formatted Edit object receives the focus, it redisplayes the contents in its raw (unformatted) form. When the Edit loses the focus, its contents are reformatted. When the user moves to a formatted Grid cell, the text remains formatted until the user presses a non-movement key or enters *in-cell* mode. The data is then redisplayed in its raw form for editing. Data in the cell is reformatted when the user moves away.

In a Grid, formatted data may be aligned vertically using the [Section 6.14](#) property as illustrated in the following example.

```
'F' □WC 'Form'
'F.G' □WC 'Grid' ('~50+?10 10p100')(0 0)(100 100)
'F.G' □WS 'FormatString' 'M<(>N<)>F12.3'
'F.G' □WS 'AlignChar' '.'
```


	A	B	C
1	{36.000}	26.000	{4.000}
2	2.000	34.000	{46.000}
3	19.000	9.000	44.000
4	27.000	{23.000}	{45.000}
5	49.000	23.000	26.000
6	{2.000}	{26.000}	{22.000}
7	1.000	2.000	{18.000}
8	{11.000}	{22.000}	42.000
9	{37.000}	{48.000}	19.000
10	{19.000}	{14.000}	2.000

6.230 FrameContextMenu Event 411

Applies To: [Section 6.227](#), [Section 6.578](#)

Description

If enabled, this event is reported when the user clicks and releases the right mouse button over the non-client area of an object, for example, the title bar in a Form.

The event message reported as the result of `OnContextMenu`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'FrameContextMenu' or 411
[3]	Y	y-position of the mouse (number)
[4]	X	x-position of the mouse (number)

For further details, see [Section 6.125](#).

6.231 FStyle Property

Applies To: [Section 6.94](#), [Section 6.190](#), [Section 6.450](#), [Section 6.472](#)

Description

This property determines how a graphics object is filled. It takes one of the following values, or, if the object has more than one component, a vector of such values.

FStyle	Effect
-1	hollow (no fill). This is the default.
0	solid fill
1	hatch fill with horizontal lines
2	hatch fill with vertical lines
3	hatch fill with diagonal lines at 135 degrees
4	hatch fill with diagonal lines at 45 degrees
5	hatch fill with horizontal and vertical lines
6	hatch fill with criss-crossing diagonal lines
str	the name of, or a ref to, a Bitmap object which is used to fill the object.

For example, to fill an object with criss-crossing diagonal lines you would specify (`'FStyle' 6`). If the object contained two components, you could fill the first one with criss-crossing diagonal lines, and the second one with a [Section 6.47](#) called 'YES', with the specification (`'FStyle' 6 'YES'`)

If the size of the [Section 6.47](#) is 8x8 APL uses a Windows "brush" to fill the object. If not, it uses "tiling". Filling with a brush is faster.

6.232 FullRowSelect Property

Applies To: [Section 6.348](#), [Section 6.629](#)

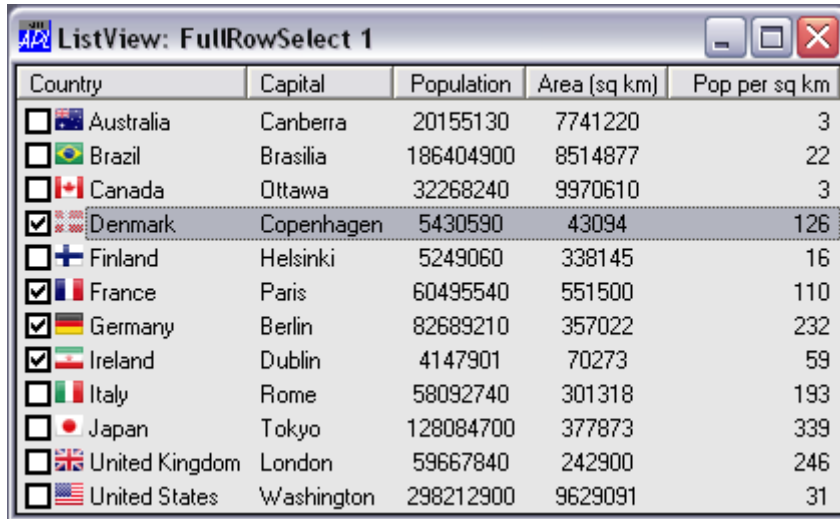
Description

The FullRowSelect property specifies whether or not the entire row is highlighted when an item in a [Section 6.348](#) or a [Section 6.629](#) is selected.

FullRowSelect is a single number with the value 0 (only the item name is highlighted) or 1 (the whole row is highlighted); the default is 0.

For a [Section 6.348](#), FullRowSelect only applies if its [Section 6.644](#) property is set to 'Report'.

The picture below illustrates the effect on the appearance of a [Section 6.348](#) object, of setting FullRowSelect to 1.



Country	Capital	Population	Area (sq km)	Pop per sq km
☐ Australia	Canberra	20155130	7741220	3
☐ Brazil	Brasilia	186404900	8514877	22
☐ Canada	Ottawa	32268240	9970610	3
☒ Denmark	Copenhagen	5430590	43094	126
☐ Finland	Helsinki	5249060	338145	16
☒ France	Paris	60495540	551500	110
☒ Germany	Berlin	82689210	357022	232
☒ Ireland	Dublin	4147901	70273	59
☐ Italy	Rome	58092740	301318	193
☐ Japan	Tokyo	128084700	377873	339
☐ United Kingdom	London	59667840	242900	246
☐ United States	Washington	298212900	9629091	31

6.233 GesturePan Event 494

Applies To

Description

This event is reported when the user touches one or two fingers on an object and drags them .

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GesturePan' or 494
[3]	Flags	integer which reports the state of the gesture
[4]	Location	2-element integer vector containing the y and x-position respectively of the point at which the gesture applies. These are reported in pixel coordinates relative to the origin (top-left corner) of the object reporting the event.
[5]	Distance	2-element integer vector containing the high and low parts (words) of a 64-bit integer that indicates the distance between the two fingers. This will be (0 0) if only one finger is used.

The Flags parameter [3] which reports the state of the Gesture, is an integer with the value 0, 1 (*GF_BEGIN*), 2 (*GF_INERTIA*), 4 (*GF_END*) or 6 (*GF_END+GF_INERTIA*) with the following meanings:

Name	Value	Description
	0	A gesture is in progress
<i>GF_BEGIN</i>	1	A gesture is starting.
<i>GF_INERTIA</i>	2	A gesture has triggered inertia.
<i>GF_END</i>	4	A gesture has finished.

The term *inertia* refers to built-in Windows processing which provides a standardised user-interface including smooth acceleration and de-acceleration of an object.

When the user first touches an object and begins to drag his finger(s), the object generates a GesturePan event with a *Flags* parameter of 1 (*GF_BEGIN*). Subsequently, if the user drags the object steadily it generates a series of GesturePan events with a *Flags* parameter of 0. When the user lifts his finger(s) away, the object generates a final GesturePan event, with a *Flags* parameter of 4 (*GF_END*).

If the user *flicks* an object, the system typically continues to generate GesturePan events after the user has ceased to touch the object. These events are generated in response to the acceleration and deceleration imparted by the *flick*, and the *Flags* parameter for these generated events will be 2 (*GF_INERTIA*) followed (for the last GesturePan event) by 6 (*GF_END+GF_INERTIA*).

No other event will be reported between the start and end of a series of GesturePan events.

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

Returning zero from the callback disables any default handling by the operating system.

6.234 GesturePressAndTap Event 497

Applies To

Description

This event is reported when the presses one finger on an object and then taps it with a second finger.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GesturePressAndTap' or 497
[3]	Flags	integer which reports the state of the gesture
[4]	Location	2-element integer vector containing the y and x-position respectively of the point midway between the two fingers. These are reported in pixel coordinates relative to the origin (top-left corner) of the object reporting the event..
[5]	Offset	3-element integer vector whose first element is (currently) 0 and whose second and third elements contain the (y,x) offset of the second finger relative to the first.

The Flags parameter [3] which reports the state of the Gesture, is an integer with the value 0, 1 (*GF_BEGIN*), or 4 (*GF_END*):

Name	Value	Description
	0	A gesture is in progress
GF_BEGIN	1	A gesture is starting.
GF_END	4	A gesture has finished.

When the user taps with his second finger, the object generates a GesturePressAndTap event with a *Flags* parameter of 1 (*GF_BEGIN*). Subsequently, until the user removes

his first finger, it generates a series of `GesturePressAndTap` events with a `Flags` parameter of 0. When the user lifts his first finger away, the object generates a final `GesturePressAndTap` event, with a `Flags` parameter of 4 (`GF_END`)

No other event will be reported between the start and end of a series of `GesturePressAndTap` events.

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

Returning zero from the callback disables any default handling by the operating system.

6.235 GestureRotate Event 495

Applies To

Description

This event is reported when the user touches two fingers on an object and twists them clockwise or anticlockwise.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GestureRotate' or 495
[3]	Flags	integer which reports the state of the gesture
[4]	Location	2-element integer vector containing the y and x-position respectively of the point midway between the two fingers. These are reported in pixel coordinates relative to the origin of the screen.
[5]	Angle	a scalar number which represents the angle of rotation of the twist measured in radians (0 → 02) from the x-axis in a counter-clockwise direction.

The `Flags` parameter [3] which reports the state of the Gesture, is an integer with the value 0, 1 (`GF_BEGIN`), or 4 (`GF_END`) with the following meanings:

Name	Value	Description
	0	A gesture is in progress
GF_BEGIN	1	A gesture is starting.
GF_END	4	A gesture has finished.

When the user first touches two fingers on an object and begins to twist, the object generates a `GestureRotate` event with a `Flags` parameter of 1 (`GF_BEGIN`). As the user continues to twist his fingers, the object generates a series of `GestureRotate` events with a `Flags` parameter of 0. When the user lifts one or both fingers away, the object generates a final `GestureRotate` event, with a `Flags` parameter of 4 (`GF_END`).

No other event will be reported between the start and end of a series of `GestureRotate` events.

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

Returning zero from the callback disables any default handling by the operating system.

6.236 GestureTwoFingerTap Event 496

Applies To

Description

This event is reported when the user taps two fingers at the same time on an object

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GestureTwoFingerTap' or 496
[3]	Flags	integer which reports the state of the gesture
[4]	Location	2-element integer vector containing the y and x-position respectively of the point midway between the two fingers. These are reported in pixel coordinates relative to the origin (top-left corner) of the object reporting the event..
[5]	Distance	2-element integer vector containing the high and low parts (words) of a 64-bit integer that indicates the distance between the two fingers.

The Flags parameter [3] which reports the state of the Gesture, is always an integer with the value 5 (*GF_BEGIN+GF_END*).

Name	Value	Description
GF_BEGIN	1	A gesture is starting.
GF_END	4	A gesture has finished.

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

Returning zero from the callback disables any default handling by the operating system.

6.237 GestureZoom Event 493

Applies To

Description

This event is reported when the user touches two fingers on an object and moves them apart or towards each other.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GestureZoom' or 493
[3]	Flags	integer which reports the state of the gesture
[4]	Location	2-element integer vector containing the y and x-position respectively of the centre point of the zoom (the point midway between the two fingers). These are reported in pixel coordinates relative to the origin (top-left corner) of the object reporting the event.
[5]	Distance	2-element integer vector containing the high and low parts (words) of a 64-bit integer that indicates the distance between the two fingers.

The Flags parameter [3] which reports the state of the Gesture, is an integer with the value 0, 1 (*GF_BEGIN*), or 4 (*GF_END*) with the following meanings:

Name	Value	Description
	0	A gesture is in progress
<i>GF_BEGIN</i>	1	A gesture is starting.
<i>GF_END</i>	4	A gesture has finished.

When the user first touches two fingers on an object and begins to move them apart or towards each other, the object generates a GestureZoom event with a *Flags* parameter of 1 (*GF_BEGIN*). As the user continues to move the fingers apart or towards each other, the object generates a series of GestureZoom events with a *Flags* parameter of 0. When the user lifts one or both fingers away, the object generates a final GestureZoom event, with a *Flags* parameter of 4 (*GF_END*).

No other event will be reported between the start and end of a series of GestureZoom events.

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

Returning zero from the callback disables any default handling by the operating system.

6.238 GetBuildID Method 992

Applies To: [Section 6.489](#)

Description

This method is used to obtain the Build ID of a Dyalog executable or the checksum of a file.

The argument to GetBuildID is $\emptyset$ or a single item as follows:

[1]	File name	character vector

The (shy) result is an 8-element character vector of hexadecimal digits that represents the Build ID.

If the argument is $\emptyset$, the build id is that of the current version of Dyalog that is running the expression.

Note that although this method is designed to uniquely identify different versions of Dyalog by its checksum, it may be used to obtain a checksum for *any* arbitrary file.

EXAMPLES

```
GetBuildID  $\emptyset$ 
38091b76
GetBuildID 'E:\DIALOG81\DIALOG.EXE'
cbf0d376
GetBuildID 'myfile'
4a29334d
```

Note that if the file does not exist, the result is 00000000.

6.239 GetCellRect Method 201

Applies To: [Section 6.263](#)

Description

This method returns the rectangle associated with a particular cell in a Grid.

The argument to GetCellRect is a 2-element vector as follows:

[1]	Row	integer
[2]	Column	integer

The result is a 2-element nested vector. The first element contains the y and x-coordinate of the top-left corner of the cell. The second element contains the height and width of the cell.

The result is reported in terms of the coordinate system of the Grid object.

6.240 GetCommandLine Method 145

Applies To: [Section 6.489](#)

Description

The GetCommandLine method returns the command line that was used to start the current Dyalog APL session or application.

The GetCommandLine method is niladic.

The result is a character vector.

EXAMPLES

```
GetCommandLine
"C:\Program Files\Dyalog\Dyalog APL-64 13.2 Unicode\dyalog.exe"

⎕←2 ⎕NQ '.' 'GetCommandLine'
"C:\Program Files\Dyalog\Dyalog APL-64 13.2 Unicode\dyalog.exe"
```

Note

GetCommandLine only works on Windows, and its use is deprecated in favour of [Section 6.241](#), which works on all platforms.

6.241 GetCommandLineArgs Method 148

Applies To: [Section 6.489](#)

Description

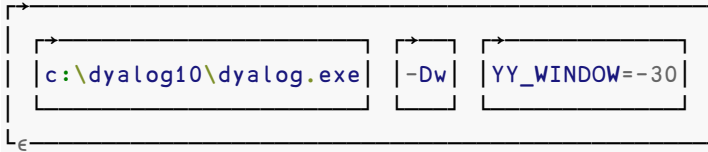
The GetCommandLineArgs method returns the command and the arguments to the command that was used to start the current Dyalog APL session or application.

The GetCommandLineArgs method is niladic.

The result is a vector of character vectors. For example:

```
GetCommandLineArgs
C:\Dyalog10\dyalog.exe -Dw YY_WINDOW=-30
```

```
DISPLAY 2 [NQ] '.' 'GetCommandLineArgs'
```



6.242 GetComment Method 222

Applies To: Section 6.263

Description

This method is used to retrieve the comment associated with a cell in a [Section 6.263](#).

The argument to `GetComment` is a 2-element array as follows:

[1]	Row	integer
[2]	Column	integer

For example, the following expression retrieves the comment associated with the cell at row 3, column 1.

```

F.C.GetComment 3 1
1 3 Hello 175 100

```

Note that to retrieve a comment associated with a row or column *title*, the appropriate element in the argument should be `~1`.

If there is no comment associated with the specified cell, the result is a scalar 1.

6.243 GetDayStates Event 266

Applies To: Section 6.58

Description

If enabled, this event is reported when a [Section 6.58](#) object requires the APL program to provide *day state* information for the range of dates it is about to display.

The [Section 6.58](#) object displays day numbers using either the normal or the bold font attribute. However, it does not store this information beyond the month or months currently displayed.

When the [Section 6.58](#) control scrolls (and potentially at other times), it generates a GetDayStates event to ask you (the APL program) to tell it which of the dates that are about to be shown, should be displayed using the bold font attribute.

If you wish any dates to be displayed using the bold font attribute, you **must** attach a callback function to this event which returns day state information in its result.

By default, all dates are displayed using the normal font attribute, so you need only do this if you want any dates highlighted in bold.

You may not disable or nullify the operation that caused GetDayStates to fire by setting the action code for the event to -1 or by returning 0 from a callback function.

The event message reported as the result of □□□, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GetDayStates' or 266
[3]	First Date	an integer (IDN)
[4]	Last Date	an integer (IDN)
[5]	Bold Dates	an integer vector of IDNs.

When the callback function is invoked, the 3rd and 4th elements of the event message contain IDNs for the first and last date in the range of dates that the [Section 6.58](#) object is about to display. The 5th element of the event message contains those IDNs *within this range of dates* that the [Section 6.58](#) control already knows are to be displayed using the bold font attribute. This will typically be empty.

The result of your callback function should be the same event message with only the 5th element modified in any way. This should contain the IDNs of the dates (within the range specified by the 3rd and 4th elements) that are to be displayed using the bold font attribute.

EXAMPLE

Suppose that you keep a variable `BOLD_DATES` in the [Section 6.58](#) object. This variable is a vector of IDN values that defines those dates that the user has somehow identified as special and that you wish to display in bold. The following callback function could be applied:

```

▼ MSG←DAYSTATES MSG;MASK;[]IO
[1]  A Callback function for the GetDayStates event
[2]  A Object (>MSG) contains a variable BOLD_DATES
[3]  A that defines ALL the IDNs that are to be displayed in bold
[4]  A We need to return only those that fall within the range
[5]  A of dates that are about to be displayed by the Calendar
[6]  []CS>MSG
[7]  []IO←1
[8]  MASK←BOLD_DATES≥3>MSG
[9]  MASK←MASK^BOLD_DATES≤4>MSG
[10] MSG[5]←<MASK/BOLD_DATES
▼

```

You may also set the font attribute for particular days by calling `GetDayStates` as a method.

For example, to set the bold attribute for IDN 36048 (11 September 1998) in a [Section 6.58](#) object called 'F.CAL1', you could execute the expression:

```
F.CAL1.GetDayStates 36048 36048 36048
```

To *clear* the bold attribute for the same day:

```
F.CAL1.GetDayStates 36048 36048 0
```

Note that the [Section 6.58](#) object will ignore any IDNs you specify that are outside the range of dates that it is currently displaying.

6.244 GetEnvironment Method 510

Applies To: [Section 6.489](#)

Description

This method is used to obtain information about one or more parameters that were specified in the Dyalog command line, your Windows registry, or defined as environment variables. If a value is defined in several places (for example, MAXWS in

the command line overriding MAXWS in the registry), GetEnvironment follows exactly the same logic as is used by Dyalog APL itself and so obtains the same value.

The argument to GetEnvironment is a single item as follows:

[1]	Parameter name(s)	see below

Parameter names is simple character vector or vector of character vectors specifying one or more parameters.

The result is a simple character vector or a vector of character vectors.

EXAMPLES

```
GetEnvironment 'MAXWS'
2G

GetEnvironment <'LOG_SIZE' 'MAXWS'
16 2G
```

Note that you may use GetEnvironment to obtain the values of your own arbitrary parameters given on the APL command line or specified as environment variables.

GetEnvironment is not supported by DYALOG.DLL because it does not use parameters.

6.245 GetEventInfo Method 551

Applies To: [Section 6.412](#), [Section 6.414](#)

Description

This method is used to obtain information about a particular event or set of events supported by a COM object.

For each event supported by a COM object, the author will have registered the data type of its result (if it has a result), a help message or description of the event (optional) and the name and data type of each of its parameters. These event parameters make up the array returned by □DQ or supplied as an argument to your callback function. The GetEventInfo method returns this information.

The argument to GetEventInfo is a single item as follows:

[1]	Event name(s)	see below

Event name(s) is a simple character vector or a vector of character vectors specifying one or more names of events supported by the object.

The result is a nested vector with one element per event name. Each element of this vector is itself a vector of 2-element character vectors. For each event, the first item describes the help message or description (if any) registered for the event and the data type of its result. Each of the remaining elements contains a parameter name and its corresponding data type.

EXAMPLE

```
CLNAME←'Microsoft Multimedia Control, Version 6.0'
'MM' □WC 'OCXClass' CLNAME

MM.EventList
Done BackClick PrevClick NextClick PlayClick ...

DISPLAY ↑MM.GetEventInfo 'Done'
```

```
graph TD
    subgraph MM_EventList [MM.EventList]
        direction LR
        subgraph TopRow
            direction LR
            A[Occurs when an MCI command ...] --> B[VT_VOID]
        end
        subgraph BottomRow
            direction LR
            C[NotifyCode] --> D[VT_PTR to VT_I2]
        end
        A --> C
        B --> D
    end
```

Note that if the event does not produce a result, the data type of the result is reported as 'VT_VOID'.

6.246 GetFocus Method 511

Applies To

Description

This method is used to obtain the name of the object that currently has the input focus.

The GetFocus method is niladic.

The result is a simple character vector. An empty result indicates that no Dyalog APL GUI object has the input focus.

See also: Section 6.247.

6.247 GetFocusObj Method 509

Applies To

Description

This method is used to obtain a ref to the object that currently has the input focus.

The GetFocusObj method is niladic.

The result is a ref. If there is no Dyalog APL GUI object with the input focus, the result is `⍬NULL`.

See also: [Section 6.246](#).

6.248 GetItemHandle Method 313

Applies To: [Section 6.629](#)

Description

This method is used to obtain the window handle of a particular item in a [Section 6.629](#) object.

The argument for GetItemHandle is a single item as follows:

[1]	Item number	Integer.

Item number is the index of the item concerned.

The result is an integer containing the window handle of the item.

6.249 GetItemPosition Method 323

Applies To: [Section 6.348](#)

Description

This method is used to obtain the position of a particular item in a [Section 6.348](#) object.

The argument for GetItemPosition is a single item as follows:

[1]	Item number	Integer

Item number is the index of the item concerned.

The result is a 2-element vector containing the position of the item.

6.250 GetItemState Method 306

Applies To: [Section 6.348](#), [Section 6.629](#)

Description

This method is used to obtain the status of a particular item in a [Section 6.348](#) or [Section 6.629](#) object.

The argument for GetItemState is a single item as follows:

[1]	Item number	Integer

Item number is the index of the item concerned. Be aware that this is index origin dependent.

The result indicates the state of the item as the sum of one or more of the following codes:

-1	Error (most likely the Item number is invalid)
1	Item has the focus
2	Item is selected
8	Item is highlighted for dropping
16	Item is displayed in bold text
32	Item is expanded
64	Item is or has been expanded
4096	Item is checked. See CheckBoxes

6.251 GetMethodInfo Method 552

Applies To: [Section 6.412](#), [Section 6.414](#)

Description

This method is used to obtain information about a particular method or set of methods supported by a COM object.

For each method supported by a COM object the author will have registered a help message or description of the method (this is in fact optional), the data type of its result (if it has a result), and the name and data type of each of the parameters that must be supplied when you invoke it. The GetMethodInfo method returns this information.

The argument to GetMethodInfo is a single item as follows:

[1]	Method name(s)	see below

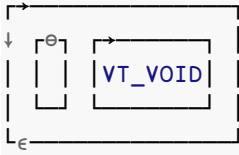
Method name(s) is a simple character vector or a vector of character vectors specifying one or more names of methods supported by the object.

The result is a nested vector with one element per method name. Each element of this vector is itself a vector of 2-element character vectors. For each method, the first item describes the help message or description (if any) registered for the method and the data type of its result. Note that if the event does not produce a result, the data type of the result is reported as 'VT_VOID'. Each of the remaining elements contains a parameter name and its corresponding data type.

EXAMPLE

```
CLNAME←'Microsoft Multimedia Control, Version 6.0'
'MM' □WC 'OCXClass' CLNAME
MM.MethodList
AboutBox Refresh OLEDrag

DISPLAY ↑ MM.GetMethodInfo 'AboutBox'
```



6.252 GetMinSize Method 275

Applies To: Section 6.58

Description

This method is used to obtain the minimum size that you must specify for a [Section 6.58](#) object for it to display a complete month.

The GetMinSize method is niladic.

The (shy) result of the method is a 2-element numeric vector containing the minimum height and width required for the object to display a complete month.

6.253 GetParentItem Method 312

Applies To: [Section 6.629](#)

Description

This method is used to obtain the index of the parent of a particular item in a [Section 6.629](#) object.

The argument for GetParentItem is a single item as follows:

[1]	Item number	Integer.

Item number is the index of the item concerned.

The result is an integer containing the index of the parent item.

6.254 GetPropertyInfo Method 550

Applies To: [Section 6.412](#), [Section 6.414](#)

Description

This method is used to obtain information about a particular property or set of properties supported by a COM object.

For each property supported by a COM object, the author will have registered the property name, its data type, and an optional help message or description of the property. GetPropertyInfo returns this information.

The argument to GetPropertyInfo is a single item as follows:

[1]	Property name(s)	see below

Property name(s) is a simple character vector or a vector of character vectors specifying one or more names of properties supported by the object.

The result is a nested vector with one element per property name. Each element of this vector is itself a 2-element vector of character vectors containing the data type and help message for the corresponding property.

EXAMPLE

```
CLNAME←'Microsoft Multimedia Control, Version 6.0'
'MM' □WC 'OCXClass' CLNAME
MM.PropList
Type DeviceType AutoEnable PrevVisible ...

DISPLAY ↑MM.GetPropertyInfo 'PrevVisible'
```

Determines if the Prev button is visible.

VT_VOID

P

VT_BOOL

If the data type of a property is VT_USERDEFINED, it means that the property may assume one of a set of values defined by a type list. In this case, the name of the type list is returned in place of the string "VT_USERDEFINED". Further information can be obtained using [Section 6.258](#) with this name as a parameter.

6.255 GetServiceState Method 92

Applies To: [Section 6.489](#)

Description

This method is used to obtain the current state of a Dyalog APL service running under Windows. See *Installation/Configuration: Apl Application As A Service*.

The GetServiceState method is niladic.

The result of the method is a 7-element numeric vector corresponding to the `SERVICE_STATUS` structure which is described in C++ as follows:

```
typedef struct _SERVICE_STATUS {
    DWORD dwServiceType;
    DWORD dwCurrentState;
    DWORD dwControlsAccepted;
    DWORD dwWin32ExitCode;
    DWORD dwServiceSpecificExitCode;
    DWORD dwCheckPoint;
    DWORD dwWaitHint;
} SERVICE_STATUS, *LPSERVICE_STATUS;
```

For further details, see the on-line documentation for `SERVICE_STATE` and the function `HashDefine` in the sample workspace `aplservice`.

6.256 GetTextSize Method 146

Applies To

Description

The `GetTextSize` method obtains the size of the bounding rectangle of a text item in a given font. The result is given in the co-ordinate system of the object in question. This method is useful for positioning Text objects.

`GetTextSize` duplicates the functionality of the `TextSize` property. It is recommended that you use `GetTextSize` instead of `TextSize` which may be removed in a future release of Dyalog APL.

The argument to `GetTextSize` is a 1 or 2-element array as follows:

[1]	Text item	character array
[2]	Font name	character vector

When you invoke `GetTextSize` you give the text item in whose size you are interested and, optionally, the name of a Font object. The text item may be a simple scalar, a vector or a matrix. If the Font is omitted, the result is given using the current font for the object in question.

EXAMPLES

```
'F'WC'Form'
F.GetTextSize'Hello World'
3.385416667 10.7421875

'FNT1' WC 'Font' 'Arial' 72
F.GetTextSize'Hello World' '#.FNT1'
18.75 65.4296875

F.Coord←'Pixel'
F.FontObj←'FNT1'
F.GetTextSize'Hello World'
16 77
```

6.257 GetTipText Event 325

Applies To: [Section 6.348](#), [Section 6.629](#)

Description

If enabled, this event is reported by a [Section 6.629](#) or [Section 6.348](#) object just before it displays a tip for a specific row.

The event message reported as the result of `□□□`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GetTipText' or 325
[3]	Item index	Integer (<code>□□□</code> dependent)
[4]	SubItem index	Integer (<code>□□□</code> dependent, currently always equal to <code>□□□</code>)
[5]	TipText	The text to be displayed.

Modifying and returning the 5th element of the argument to the callback function allows the application to change the displayed tip.

The text can be set to a character array of rank 2 or less.

The default processing for the event is to display the default tip (if there is one).

The associated callback is run **immediately** while the windows notification is still on the stack. See *Interfaces: High Priority Callbacks*.

6.258 GetTypeInfo Method 553

Applies To: [Section 6.412](#), [Section 6.414](#)

Description

This method is used to obtain information about a type list supported by a COM object.

The argument to GetTypeInfo is a single item as follows:

[1]	Type List name(s)	see below
[2]	Value	(usually) numeric
[3]	Description	character vector

Type List name(s) is a simple character vector or a vector of character vectors specifying one or more names of type lists supported by the object.

The result is a nested vector with one element per [Section 6.633](#). Each element of this vector is itself a 3-element vector of character vectors made up as follows:

[1]	Name of Constant	character vector
[2]	Value	(usually) numeric
[3]	Description	character vector

6.259 GetVisibleRange Method 262

Applies To: [Section 6.58](#)

Description

This method is used to obtain the range of dates that is currently visible in a [Section 6.58](#) object.

The GetVisibleRange method is niladic.

The result is a 2-element integer vector containing the first and last dates currently displayed by the object, reported as [Section 8.5](#).

6.260 GetZoomLevel Method 838

Applies To: [Section 6.307](#)

Description

This method is used to retrieve the current CEF ZoomLevel of the HTMLRenderer.

See [Section 6.542](#).

6.261 GotFocus Event 40

Applies To

Description

If enabled, this event is generated when the user has moved the keyboard focus to a new object by clicking the left mouse button, pressing TAB, or using a cursor key.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GotFocus ' or 40
[3]	Object name	character vector (name of object which previously had the focus)

The third element (object name) is empty if the focus was obtained from another application window.

The GotFocus event is generated **after** the focus has changed. The default processing is therefore to take no action. However, if you disable the event by setting its action code to -1, or inhibit it by returning a 0 from your callback function, the focus is automatically restored to the object (or external application) that had lost it.

6.262 GreetBitmap Method 138

Applies To: [Section 6.489](#)

Description

This method is used to display or remove a bitmap, typically during initialisation of a Dyalog runtime application.

The argument to GreetBitmap is $\emptyset$ or a 2 element vector as follows:

[1]	Display	0 = off, 1 = on.
[2]	Bitmap file name	Character vector.

If the argument is $\emptyset$, the bitmap is removed.

The image can also be displayed initially by setting parameter: **greet_bitmap** on the command line. For example:

```
c:\myapp\dyalogrt greet_bitmap=mylogo myws
```

The image is displayed until either an untrapped error occurs, causing the interpreter to (attempt to) display the session window, or the GreetBitmap method is called.

6.263 Grid Object

Purpose: Spreadsheet object for displaying and editing data.

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm

Children Bitmap BrowseBox Button Circle ColorButton Combo Cursor DateTimePicker Edit Ellipse FileBox Font Icon Image Label Marker Menu MsgBox OCXClass Poly Rect Spinner Text Timer TrackBar

Properties Type Values Posn Size FCol BCol Coord Border Active Visible Event VScroll HScroll SellItems Sizeable Dragable FontObj CursorObj AutoConf Index YRange XRange Data Attach TextSize EdgeStyle Handle Hint HintObj Tip TipObj FormatString RowTitles ColTitles CurCell TitleWidth CellHeights CellWidths TitleHeight CellFonts Input CellTypes AutoExpand CellSelect ResizeRows ResizeCols ResizeRowTitles ResizeColTitles ClipCells InputModeKey InputMode GridFCol GridBCol ShowInput CellSet RowTitleFCol ColTitleFCol RowTitleDepth ColTitleDepth RowTitleAlign ColTitleAlign OverflowChar AlignChar GridLineFCol GridLineWidth RowLineTypes ColLineTypes EnterReadOnlyCells RowTitleBCol ColTitleBCol RowTreeDepth RowTreeStyle RowTreeImages ColSortImages SelectionColor SelectionColorAlpha SelectionBorderWidth HighlightHeaders Translate Accelerator AcceptFiles KeepOnClose Redraw InputProperties TabIndex AlwaysShowSelection AlwaysShowBorder RowHiddenDepth CornerTitleBCol MethodList ChildList EventList PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj
 DelRow DelCol SetCellType RowChange ColChange Undo SetCellSet
 RowSetVisibleDepth ColSorted DuplicateRow DuplicateColumn CellFromPoint
 GetCellRect LockRows LockColumns AddComment DelComment GetComment

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles
 DropObjects Expose Help KeyPress GotFocus LostFocus MouseEnter MouseLeave
 GridKeyPress CellChange CellMove AddRow AddCol CellError CellOver CellDown CellUp
 CellDbClick CellChanged GridSelect Expanding Retracting SetRowSize SetColSize
 GridCut GridCopy GridPaste GridDelete GridPasteError GridDropSel GridCopyError
 IndexChanged ShowComment HideComment ClickComment Select

Description

The [Section 6.642](#) property is a matrix whose elements are displayed in the cells of the Grid. An element (and therefore a cell) may contain a single number, a single character, a character vector or a character matrix.

The [Section 6.77](#) property specifies the height of each of the rows of the spreadsheet. It may be a single value which applies to all rows, or a vector with one element per row. The [Section 6.84](#) property determines the width of each column of the spreadsheet. It too may be a single value or a vector with one element per column.

The [Section 6.500](#) property is either an empty character vector (the default) or a vector of character vectors that specify row titles displayed to the left of the cells in the Grid. If [Section 6.500](#) is not specified, the Grid labels each row with its row number. The [Section 6.118](#) property is similar and is used to specify column headings. If [Section 6.118](#) is not specified, the Grid displays "standard" spreadsheet column headings A-Z, then AA-AZ and so forth.

The [Section 6.618](#) property specifies height of the column headers. If this is set to 0, the column titles will not be displayed. Similarly, the [Section 6.619](#) property specifies the width of the row titles and again a value of zero disables the row titles.

The [Section 6.13](#) property specifies the background colour of the rectangle in the top-left corner of the Grid.

The [Section 6.225](#) property may be used to specify the font to be used for the Grid as a whole, including the titles. The [Section 6.75](#) property may be used to specify fonts for individual cells.

The [Section 6.205](#) and [Section 6.45](#) properties may each specify a single colour for the Grid as a whole, or may specify a vector of colours whose elements are mapped to individual cells through the [Section 6.82](#) property.

The [Section 6.75](#) property is either a character vector or a vector of character vectors that specifies the name of a single font object to be used for all cells in the Grid, or a vector of character vectors that specifies a set of font objects that are mapped to individual cells through the [Section 6.82](#) property.

The [Section 6.321](#) property is a character vector that specifies the name of an object which is to be associated with every cell in the Grid, or a vector of names whose elements are mapped to individual cells through the [Section 6.82](#) property. These objects may be of type [Section 6.55](#), [Section 6.108](#), [Section 6.121](#), [Section 6.185](#), [Section 6.341](#), [Section 6.563](#) or [Section 6.625](#). In addition, the [Section 6.321](#) property may specify instances of [Section 6.412](#) objects (ActiveX controls) and [Section 6.408](#) objects (.NET classes).

If the [Section 6.321](#) property is empty (the default) the user may browse the data in the spreadsheet but may not alter it. Furthermore, no feedback is provided as to which is the current cell. If the [Section 6.321](#) property specifies the name of an object that is the child of the Grid itself, this object *floats* from cell to cell as the user moves around the spreadsheet, and the current cell is identified by its presence.

If the [Section 6.321](#) property specifies the name of an external object (that is, an object that is *not* a child of the Grid), the contents of the current cell are copied into that object as the user moves around the spreadsheet. In addition, the current cell is identified by a thick border. In either case, the associated object is used to impose formatting and validation.

If the [Section 6.321](#) property specifies the name of a [Section 6.341](#) object, that object is used to impose formatting, but the data is protected and may not be changed. If the [Section 6.341](#) is a child of the Grid, it moves from cell to cell, and its characteristics ([Section 6.49](#), [Section 6.205](#), [Section 6.45](#) and [Section 6.225](#)) can be used to identify the current cell. If the [Section 6.341](#) is an external one, no visual feedback is provided; even though the current cell (reflected by the [Section 6.133](#) property) changes as the user moves around the Grid.

If the Input property specifies one or more instances of OCXClass objects (ActiveX controls) and [Section 6.408](#) objects (.NET classes), the [Section 6.324](#) property is used to map the [Section 6.642](#) property of the Grid to specific properties of the external object.

The [Section 6.82](#) property is either an empty numeric matrix (the default) or an integer matrix of the same shape as [Section 6.642](#). If specified, each element of [Section 6.82](#) determines the index into various properties, including the [Section 6.205](#), [Section 6.45](#), [Section 6.75](#) and [Section 6.321](#) properties, to be used for the corresponding cell. For example, if an element in [Section 6.82](#) is 3, the 3rd element of [Section 6.205](#) is used for

the foreground colour of the corresponding cell, the 3rd element of [Section 6.45](#) specifies the background colour, and so forth.

The [Section 6.133](#) property may be used to set or query the current cell. The current cell is the cell which the user has picked by clicking the mouse over it or by using the cursor keys. [Section 6.133](#) is a 2-element vector containing the current cell's row number and column number respectively and is $\square$ IO dependent. The [Section 6.319](#) property specifies the row and column number of the cell in the top-left corner of the Grid. It too is $\square$ IO dependent.

The [Section 6.36](#) property is a 2-element Boolean vector which specifies whether (1) or not (0) new rows and columns are added when the user presses the corresponding cursor key when at the end of the block of cells. Its default value is (0 0).

The Grid object reports a [Section 6.73](#) event when the user depresses a mouse button over a cell. The event message contains the row and column address of the cell in question which is $\square$ IO dependent. It also reports a similar [Section 6.83](#) event when the mouse button is released and a [Section 6.72](#) event when it is double-clicked. The number of the mouse button and the state of the shift keys are also reported.

When the user moves to another cell, the Grid object reports a [Section 6.78](#) event. This simply reports the address of the new cell and may be used to take some appropriate action when a particular cell is picked. If the user alters the data in a cell and then attempts to move to another, the Grid reports a [Section 6.70](#) event. This can be used to perform validation.

Alternatively, the Grid may report a [Section 6.71](#) event which occurs after the [Section 6.642](#) property has been updated with the new cell contents. This may be used to perform immediate recalculation.

The [Section 6.12](#) event is generated if the current cell is in the last row of the Grid and the user presses Cursor Down. By default, this operation adds a new row to the Grid, but you can attach a callback to the [Section 6.12](#) and selectively disable this default action if required. The [Section 6.9](#) event works in a similar manner for columns. Although the user has no direct means of *inserting* a row or column, your application can do this by calling [Section 6.12](#) or [Section 6.9](#) as a method on the Grid object. Typically this would be done in response to the user selecting a [Section 6.384](#) or pressing a [Section 6.55](#).

The [Section 6.105](#), [Section 6.491](#), [Section 6.155](#), [Section 6.150](#) and [Section 6.635](#) methods allow your application to perform these corresponding operations.

The Grid object maintains a buffer of the most recent 8 changes made by the user since the [Section 6.642](#) property was last set by *Dyalog APL Language: Wc* or *Dyalog APL*

Language: Ws. Your application can restore these changes one by one by calling the [Section 6.635](#) method. The [Section 6.635](#) method restores the most recent change made by the user and removes that change from the undo stack. It is therefore not possible to "undo an undo".

The Grid supports the selection of a block or blocks of cells using the mouse and/or the keyboard. The ability to select a range of cells is determined by the [Section 6.80](#) property. When the user performs a selection, the Grid generates a [Section 6.277](#) event. The range of cells currently selected is given by the [Section 6.519](#) property.

If a block of cells has been selected, the user may delete the contents, and cut or copy the contents of the cells to the clipboard by pressing Delete, Shift+Delete or Ctrl+Insert respectively. These operations also generate [Section 6.268](#), [Section 6.267](#) and [Section 6.265](#) events which you can selectively disable using a callback function. You can also perform these operations under program control by calling them as methods.

If more than one block of cells has been selected, these operations are honoured only if the blocks begin and end on the same rows or begin and end on the same columns. If so, the data placed in the clipboard is the result of joining the blocks horizontally or vertically as appropriate.

The user may paste data from the clipboard into a Grid by pressing Shift+Insert. Data is pasted into the currently selected block of cells, or, if there is no selection, data is pasted starting at the current cell (`CurCell`). The operation also generates a [Section 6.275](#) event, and, if the operation cannot proceed, a [Section 6.276](#) event.

If you move the mouse pointer over any of the four edges of a selected block of cells, the cursor changes to an arrow. You may now click and drag the border of the selected cells with the mouse.

If you press the Ctrl key at the same time, the contents of the selected cells are *copied* to the new location, replacing the values in the block of cells onto which they are dropped. Otherwise, the operation is treated as a *move* and the original block of cells is emptied. This operation also generates a [Section 6.269](#) event. You may only move or copy a *single* block of cells in this way.

The user may be permitted to resize the rows and/or columns of a Grid. This is controlled by the [Section 6.483](#) and [Section 6.481](#) properties whose default values are 0. To allow the user to resize, set either or both to 1. You can also specify a Boolean vector to allow specific rows/columns to be resized while others are fixed. Two additional properties named [Section 6.484](#) and [Section 6.482](#) determine whether or not the user may alter the width of the row titles and the height of the column titles.

If resizable, the cursor changes to a double-heads arrow when the user moves the mouse pointer over the lines between the row and/or column titles. The user may click and drag with the mouse to the desired size. The user may also double-click. This causes the row or column to be resized to fit the data. Both operations generate a [Section 6.527](#), or [Section 6.536](#) event.

When you edit data in a Grid, the editing behaviour and the action of the cursor movement keys is determined by the [Section 6.322](#) and [Section 6.323](#) properties.

The [Section 6.270](#) property specifies the colour of all the grid lines. Alternatively, the [Section 6.272](#), [Section 6.274](#), [Section 6.493](#) and [Section 6.107](#) properties may specify the appearance for individual grid lines.

The [Section 6.264](#) property specifies the colour used to fill the area between the end of the last column of data and the right edge of the Grid and between the bottom row of data and the bottom edge of the Grid.

The [Section 6.499](#) and [Section 6.117](#) properties specify the colours to be used for the row and column titles respectively.

The [Section 6.100](#) property determines whether or not the Grid displays partial cells. The default is 1. If you set [Section 6.100](#) to 0, the Grid displays only complete cells and automatically fills the space between the last visible cell and the edge of the Grid with the [Section 6.264](#) colour.

The [Section 6.81](#) property is a Boolean array that marks which cells are set (that is, have values) and which are empty. This allows you to edit large numeric matrices which contain empty cells without a severe workspace penalty.

The [Section 6.303](#) and [Section 6.646](#) properties specify whether or not horizontal and vertical scrollbars are displayed. Either property may be given the value -3 which forces the corresponding scrollbar to appear always. [Section 6.646](#) and [Section 6.303](#) may only be set when the object is created and may not subsequently be changed.

The Grid object supports *comments* in a manner that is consistent with the way that comments are handled by Microsoft Excel. If a comment is associated with a cell, a small red triangle is displayed in its top right corner. When the user rests the mouse pointer over a commented cell, the comment is displayed as a pop-up with an arrow pointing back to the cell to which it refers. The comment disappears when the mouse pointer is moved away. This is referred to as *tip* behaviour. Comments may also be associated with row and column titles.

Grid comments are managed by a set of methods, namely [Section 6.10](#), [Section 6.151](#), [Section 6.242](#), [Section 6.545](#), [Section 6.297](#) and [Section 6.98](#).

You may lock individual rows and columns using the [Section 6.358](#) and [Section 6.357](#) methods. This facility is however not supported in combination with hierarchical rows and/or columns which are specified by [Section 6.498](#) and [Section 6.116](#).

The Grid can display a *TreeView like* interface on the Row titles. In this mode, the Grid automatically shows and hides row of data as the end user expands and contracts nodes of the tree.

The [Section 6.501](#) property is used to specify the depth of rows in the Grid.

The appearance of the tree is determined by the [Section 6.503](#) property.

User defined bitmaps can be used instead of the default Images by setting the [Section 6.502](#) property.

The Grid generates [Section 6.201](#) and [Section 6.487](#) events when the user interacts with the tree.

The [Section 6.495](#) method can be used to set the visible depth of the tree.

6.264 GridBCol Property

Applies To: [Section 6.263](#)

Description

This property specifies the colour used to fill the area between the end of the last column of data and the right edge of the [Section 6.263](#) and between the bottom row of data and the bottom edge of the [Section 6.263](#).

GridBCol may be a 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or it may be a scalar that defines a standard Windows colour element (see [Section 6.45](#) for details). Its default value is 0 which obtains the colour defined for Window Background.

6.265 GridCopy Event 191

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported when the user presses Ctrl+Insert and there are selected cells in the [Section 6.263](#). The default action of the event is to copy the

contents of the selected block of cells to the clipboard. You may disable this effect entirely by setting the action code of the event to `⍒1`. You may also disable the copy operation by returning 0 from a callback function.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'GridCopy' or 191
[3]	Start	2-element integer vector or matrix containing the row, column address(es) of the top left cell(s) in the selected block(s)
[4]	End	2-element integer vector or matrix containing the row, column address(es) of the bottom right cell(s) in the selected block
[5]	Values	2-element nested vector. The first element is a matrix containing the values of the selected block(s) of cells. This is the data that will be copied to the clipboard. The second element is a Boolean matrix containing the values of the <u>CellSet</u> property for the selected block(s) of cells.

Note that the values of Start and End are sensitive to the index origin, `⍋IO`.

If more than one block of cells is selected, Start and End are matrices whose rows identify the start and end cells of each of the selected blocks, and Data is the contents of the selected blocks catenated along the appropriate dimension according to their relative positions in the Grid.

You may copy cells under program control by calling GridCopy as a method.

To copy a specific block of cells to the clipboard whether or not they are selected, you must specify the Start and End parameters. For example, the following expression will copy the 3x3 block of cells in the top-left of the Grid (`⍋IO` is 1) to the clipboard:

```
Gridname.GridCopy (1 1) (3 3)
```

If you omit these parameters, the currently selected block of cells will be copied to the clipboard. If no cells are selected, the entire contents of the Grid will be copied. That is:

```
Gridname.GridCopy 0
```

The data copied to the clipboard is registered in Dyalog (APL internal), Wk3 (Lotus), XlTable (Excel) and tab/new-line delimited text formats.

6.266 GridCopyError Event 196

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported when the user presses Ctrl+Insert and there is more than one block of selected cells in the [Section 6.263](#) and the blocks are non-conformable. The default action of the event is to generate a Beep. Setting the action code of this event to `⍒1`, or returning a 0 from a callback function attached to it, disables the Beep.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'GridCopyError' or 196
[3]		Zilde
[4]		Zilde
[5]	Start	A 2-column integer matrix whose rows identify the address of the first cell (row, column) of each of the selected blocks of cells.
[6]	End	A 2-column integer matrix whose rows identify the address of the last cell (row, column) of each of the selected blocks of cells.

Note that the values of Start and End are sensitive to the index origin, `⍋IO`.

6.267 GridCut Event 190

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported when the user presses Shift+Delete and there are selected cells in the [Section 6.263](#). The default action of the event is to copy the contents of the selected block(s) of cells to the clipboard and then to empty the selected cells. You may disable this effect entirely by setting the action code of the event to `⍒1`. You may also disable the cut operation by returning 0 from a callback function.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'GridCut ' or 190
[3]	Start	2-element integer vector or matrix containing the row, column address(es) of the top left cell(s) in the selected block(s)
[4]	End	2-element integer vector or matrix containing the row, column address(es) of the bottom right cell in the selected block
[5]	Data	2-element nested vector. The first element is a matrix containing the values of the selected block(s) of cells. This is the data that will be copied to the clipboard. The second element is a Boolean matrix containing the values of the <u>CellSet</u> property for the selected block(s) of cells.

Note that the values of Start and End are sensitive to the index origin, $\square$ IO.

If more than one block of cells is selected, Start and End are matrices whose rows identify the start and end cells of each of the selected blocks, and Data is the contents of the selected blocks catenated along the appropriate dimension according to their relative positions in the Grid.

The data copied to the clipboard is registered in Dyalog (APL internal), Wk3 (Lotus), XlTable (Excel) and tab/new-line delimited text formats.

6.268 GridDelete Event 193

Applies To: Section 6.263

Description

If enabled, this event is reported when the user presses Delete and there are selected cells in the Section 6.263. The default action of the event is to empty the selected cells. You may disable this effect entirely by setting the action code of the event to $\bar{1}$. You may also disable the delete operation by returning 0 from a callback function.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 4-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'GridDelete' or 193
[3]	Start	2-element integer vector or matrix containing the row, column address(es) of the top left cell(s) in the selected block(s)
[4]	End	2-element integer vector or matrix containing the row, column address(es) of the bottom right cell(s) in the selected block(s)

Note that the values of Start and End are sensitive to the index origin, $\square$ IO.

If more than one block of cells is selected, Start and End are matrices whose rows identify the start and end cells of each of the selected blocks.

6.269 GridDropSel Event 195

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported when the user drag/drops a selected block of cells in the [Section 6.263](#). The default action is that the contents of the selected cells replace the values in the block of cells onto which they are dropped and this block now becomes selected.

You may disable the drag/drop facility entirely by setting the action code of the event to $\neg$ 1. You may also disable an individual drag/drop operation by returning 0 from a callback function.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GridDropSel' or 195
[3]	Start	2-element integer vector containing the row, column address of the top left cell in the selected block
[4]	Size	2-element integer vector containing the number of rows and columns in the selected block
[5]	Target	2-element integer vector containing the row/column address of the top left cell onto which the selected block is being dropped
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down
[7]	Undo flag	0 or 1
[8]	Values	Matrix containing the values of the selected block of cells. This is the data that will replace the values in the target cells.
[9]	CellSet flags	Boolean Matrix containing the values of the <u>CellSet</u> property for the selected block of cells. This will replace the values of the <u>CellSet</u> property of the target cells.

The shift state in element 6 is intended to allow the APL programmer to implement an *insert* operation instead of a *copy* or *move* operation if required.

You may copy the contents of one block of cells to another by calling GridDropSel as a method. If so, you need only specify the *Start*, *Size* and *Target* parameters. Note that the result block becomes selected.

The Undo flag is always 1 if the event was generated by the user.

6.270 GridFCol Property

Applies To: Section 6.263

Description

The GridFCol property specifies the colour of the grid lines in a Section 6.263 object.

GridFCol may be a 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or it may be a scalar

that defines a standard Windows colour element (see [Section 6.45](#) for details). Its default value is 0 which obtains the colour defined for Window text.

The grid lines may be removed by setting GridFCol to the same colour as the background colour of the cells, which is defined by [Section 6.45](#).

Finer control of the colour of the grid lines can be achieved by using [Section 6.272](#) instead.

Unlike [Section 6.272](#), the value of GridFCol is ignored when *Native Look and Feel* is enabled; the colour is taken from the current theme.

6.271 GridKeyPress Event 24

Applies To: [Section 6.263](#)

Description

If enabled, this event is generated when the user presses and releases a key in a [Section 6.263](#) cell.

The GridKeyPress is reported on the [Section 6.263](#), after the [Section 6.340](#) event, which is reported on the [Section 6.321](#) object associated with the current cell.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GridKeyPress' or 24
[3]	Input Code	character scalar or vector
[4]	ASCII code	integer scalar
[5]	Key Number	integer scalar
[6]	Shift State	integer scalar
[7]	Input Object	ref or character vector

For a full description of elements [3-6], see [Section 6.340](#) event.

The 7th element of the event message contains either a reference to, or the name of the Input object associated with the current cell and on which the corresponding [Section 6.340](#) event has been reported.

The default action of the GridKeyPress event is to pass a [Section 6.340](#) event message back to the appropriate Input object to be actioned. The GridKeyPress is reported on the Grid, after the [Section 6.340](#) event on the Input object. If a callback on the Input object's [Section 6.340](#) event returns 0 or if a callback on GridKeyPress returns 0, the keystroke will be ignored.

6.272 GridLineFCol Property

Applies To: [Section 6.263](#)

Description

The GridLineFCol property specifies the colours of the grid lines in a [Section 6.263](#) object. GridLineFCol should be used if different coloured grid lines are required. If all the grid lines are the same colour, use [Section 6.270](#).

GridLineFCol may be a scalar or a vector. Each item may be a 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or a scalar that defines a standard Windows colour element (see [Section 6.45](#) for details). Note that a single RGB triplet must be enclosed.

The default value of GridLineFCol is an empty numeric vector ($\emptyset$). If so, all the grid lines are drawn using the single colour specified by [Section 6.270](#).

Unlike [Section 6.264](#), setting GridLineFCol overrides the colour which would be used if *Native Look and Feel* was enabled.

Elements of GridLineFCol are allocated to individual grid lines via the [Section 6.493](#) and [Section 6.107](#) properties.

See also: [Section 6.274](#).

6.273 GridLines Property

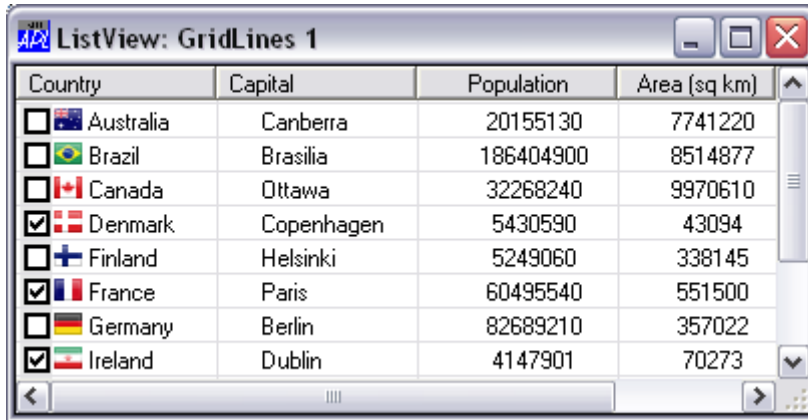
Applies To: [Section 6.348](#)

Description

The GridLines property specifies whether or not lines are displayed between items in a [Section 6.348](#) object. GridLines applies only if the value of the [Section 6.644](#) property is 'Report'.

GridLines is a single number with the value 0 (no lines are displayed) or 1 (lines are displayed); the default is 0.

The picture below illustrates the effect on the appearance of a [Section 6.348](#) object, of setting GridLines to 1.



6.274 GridLineWidth Property

Applies To: [Section 6.263](#)

Description

The GridLineWidth property specifies the widths in pixels of the grid lines in a [Section 6.263](#) object.

GridLineWidth may be an integer scalar or a vector. Its default value is an empty numeric vector ($\emptyset$). If so, grid lines are drawn 1-pixel wide.

Grid lines are always displayed so that 1 pixel is drawn *within* the cell. If the width is greater than 1 pixel, the additional pixels are drawn *between* the cells.

If an element of GridLineWidth is 0, the corresponding grid lines are not drawn.

Elements of GridLineWidth are allocated to individual grid lines via the [Section 6.493](#) and [Section 6.107](#) properties.

See also: [Section 6.272](#).

6.275 GridPaste Event 192

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported when the user presses Shift+Insert and there is data in the clipboard that is in a suitable format for the [Section 6.263](#). The default action of the event is to copy the contents of the clipboard into the currently selected block of cells, or, if no cells are selected, into the block of cells starting at the current cell (CurCell). Note that if there is a selected range of cells and the shape of the data being pasted does not exactly match the size of the selected range, the system generates a [Section 6.276](#) event in addition to the GridPaste event.

You may disable the paste facility entirely by setting the action code of the event to -1. You may also disable an individual paste operation by returning 0 from a callback function.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GridPaste' or 192
[3]	Values	New values (taken from the clipboard) which are to replace the existing values of the block of cells defined by Start and End.
[4]	CellSet flags	Boolean Matrix containing the new values of the CellSet property for the block of cells defined by Start and End.
[5]	Start	2-element integer vector containing the row, column address of the top left cell the selected block. If there is no selection, this is the address of the current cell(CurCell).
[6]	End	2-element integer vector containing the row, column address of the bottom right cell in the selected block. If there is no selection, this is the address of the bottom right cell of the block starting at the current cell that will be overwritten

You can replace the contents of a contiguous block of cells with the data in the clipboard, or with an arbitrary matrix of values, by calling GridPaste as a method.

If you call GridPaste with an argument of 0, the data is taken from the clipboard; otherwise the data to be pasted is specified by the *Values* and *CellSet flags*

parameters.. If you omit *Start*, data is pasted into the currently selected range of cells. If there are no cells selected, data is pasted starting at the current cell (*CurCell*). In either case, the block of replaced cells becomes selected.

6.276 GridPasteError Event 194

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported when the user presses Shift+Insert and there is data in the clipboard, but the system is unable to paste the data into the [Section 6.263](#). This occurs if there is a currently selected block of cells whose shape does not match the shape of the data in the clipboard. It also occurs if there is no selected block of cells, and pasting the data in starting at the current cell (*CurCell*) would overflow the [Section 6.263](#). Setting the action code of this event to `-1`, or returning a 0 from a callback function attached to it, has no effect.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'GridPasteError' or 194
[3]	Values	Contents of the clipboard.
[4]	CellSet flags	Boolean array indicating which elements of the clipboard data are empty.
[5]	Start	2-element integer vector containing the row, column address of the top left cell in the selected block. If there is no selection, this is the address of the current cell (<i>CurCell</i>).
[6]	End	2-element integer vector containing the row, column address of the bottom right cell in the selected block. If there is no selection, this is the address of the bottom right cell of the block starting at the current cell that will be overwritten
[7]	Error Number	4 (RANK ERROR) or 5 (LENGTH ERROR)

6.277 GridSelect Event 165

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported when the user performs or cancels the selection of a block of cells in a [Section 6.263](#) object. This event is reported after the selection has changed. Setting its action code to `~1` has no effect and the result of a callback function cannot be used to alter the selection that has been made. You may however control the user's ability to make selections using the [Section 6.80](#) property.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'GridSelect' or 165
[3]	Start	2-element integer vector or matrix containing the row, column address(es) of the top left cell(s) in the selected block(s)
[4]	End	2-element integer vector or matrix containing the row, column address(es) of the bottom right cell(s) in the selected block

Note that the values of Start and End are sensitive to the index origin, `⎕IO`.

If the selection is made with the mouse, the GridSelect event is reported when the left mouse button is released. If the selection is made using the cursor keys, the GridSelect event is reported when the Shift key is released.

The GridSelect event is also generated when the current selection is cancelled by clicking on a cell with the mouse or by pressing a cursor key.

6.278 GripperMode Property

Applies To: [Section 6.126](#)

Description

The GripperMode property specifies whether or not the [Section 6.126](#) has a gripper bar which is used to reposition and resize the [Section 6.126](#) within its parent [Section 6.127](#).

GripperMode is a character vector with the value 'Always' (the default), 'Never' or 'Auto'.

If GripperMode is 'Always', the [Section 6.126](#) displays a gripper bar even if it is the only [Section 6.126](#) in the [Section 6.127](#).

If GripperMode is 'Never', the [Section 6.126](#) does not have a gripper bar and may not be directly repositioned or resized by the user.

If GripperMode is 'Auto', the [Section 6.126](#) displays a gripper bar only if there are other [Section 6.126](#) in the same [Section 6.127](#).

6.279 Group Object

Purpose: This object is used to group a related set of controls together visually, and to impose "radio-button" behaviour.

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm ToolBar ToolControl

Children Animation Bitmap Button ButtonEdit Calendar Circle ColorButton Combo ComboEx Cursor DateTimePicker Edit Ellipse Font Grid Group HTMLRenderer Image ImageList Label List ListView Locator Marker Metafile Poly ProgressBar Rect RichEdit Scroll SM Spinner Splitter Static SubForm Text Timer TipField TrackBar TreeView UpDown

Properties Type Caption Posn Size Coord Border Active Visible Event Sizeable Draggable FontObj FCol BCol Picture CursorObj AutoConf YRange XRange Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ChooseFont

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap Select

Description

A Group is displayed as an empty box with a border around it whose appearance is defined by the [Section 6.184](#) property. The [Section 6.66](#) property defines a string of text that is displayed in the top left border. The default value is an empty vector.

A Group will be resized if its parent [Section 6.227](#) or Group is resized. It can also be resized directly by the user if its [Section 6.557](#) property is set to 1. By default, when a Group is resized, it automatically adjusts the size and position of its children to maintain the same proportions within it as before. The resizing of a Group and its children can be controlled using the [Section 6.35](#) property or by enabling the [Section 6.123](#) event (31).

6.280 HAlign Property

Applies To: [Section 6.605](#)

Description

This property determines the horizontal alignment of text in a [Section 6.605](#) object. It is either a single integer value, or, if the [Section 6.605](#) object contains several components, a corresponding vector of such values.

These may be:

0	left aligned	the left edge of the bounding box of the text is aligned on the x-coordinate specified by the Points property.
1	centre aligned	the centre of the bounding box of the text is aligned on the x-coordinate specified by the Points property.
2	right aligned	the right edge of the bounding box of the text is aligned on the x-coordinate specified by the Points property.

6.281 Handle Property

Applies To

Description

This is a read-only property that reports the *handle* associated with an object. For a visual object, such as a [Section 6.227](#) or a [Section 6.55](#), this is the window handle. For a [Section 6.455](#), it is the *printer device context*.

This handle allows you to access the corresponding object directly with Windows API functions via `HWND`. This facility must be used with care and the responsibility for its behaviour is entirely yours. Do NOT use it to delete an object. This will cause APL to crash.

An example of the use of the Handle property is to set tab stops in a [Section 6.346](#) object. This is illustrated by the following function:

```

▼ obj TABSTOPS stops;I;LB_SETTABSTOPS;SetTabStops;sink;args
[1]  A Sets the tabstops in the List Box OBJ to be at
[2]  A stops Horizontal Dialog units.
[3]  A Sends LB_SETTABSTOPS (402) to the List Box
[4]  A See Windows SDK for Details.
[5]
[6]  I←obj □WG'Items'
[7]
[8]  LB_SETTABSTOPS←402
[9]  'SetTabStops'□NA'U4 USER32|SendMessageA U4 U4 U4 <U4[]'
[10]
[11]  args←(obj □WG'Handle') LB_SETTABSTOPS (p,stops)(,stops)
[12]  sink←SetTabStops args
[13]
[14]  obj □WS'Items'I
▼

```

6.282 HasApply Property

Applies To: [Section 6.463](#)

Description

The HasApply property is a Boolean value that specifies whether or not a [Section 6.463](#) has an Apply button. Its default value is 1. Note that an Apply button is only actually used if Style is 'Standard'.

6.283 HasButtons Property

Applies To: [Section 6.629](#)

Description

The HasButtons property is a Boolean value and specifies whether or not buttons are shown in a [Section 6.629](#) object. If HasButtons is 1 (the default) a square button is displayed to the left of each parent item label. If the item is expanded (that is, its children are visible) the button contains a minus sign. If the item is not expanded, (that is, its children are hidden) the button contains a plus sign. The user can cause a parent item to expand or collapse by clicking this button.

6.284 HasCheckBox Property

Applies To: [Section 6.143](#)

Description

Specifies whether or not a checkbox is displayed alongside the value in a DateTimePicker.

HasCheckBox is a single number with the value 0 (the default) or 1. If HasCheckBox is 1, the user may set or clear the checkbox to indicate whether or not the date/time displayed in the object is to apply.

If the checkbox is not set, the [Section 6.143](#) is considered to be *empty* (the contents will be grayed out) and the value returned by the [Section 6.141](#) property is *zilde*. Note that HasCheckBox may only be set when the object is created.

6.285 HasClearButton Property

Applies To: [Section 6.56](#), [Section 6.121](#), [Section 6.122](#), [Section 6.185](#)

Description

Specifies whether or not a

✕

button is displayed in the right-hand end of an edit box. Clicking this button clears the text from the field.

Note

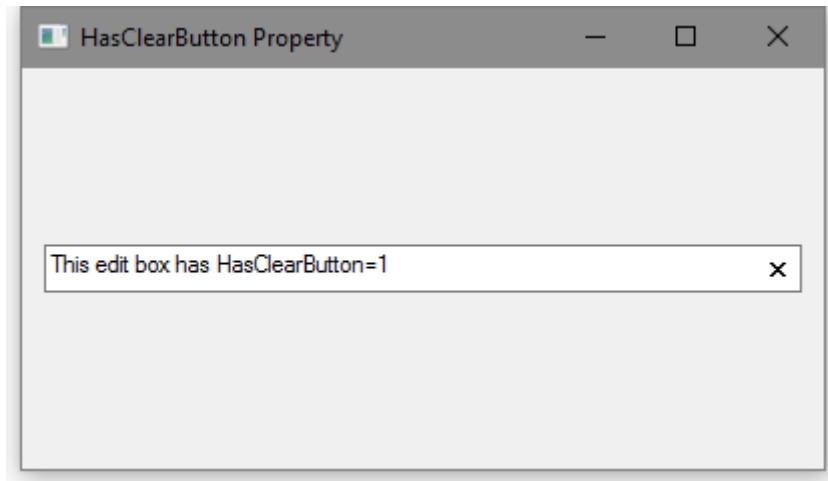
This feature only applies if [Section 8.6](#) is enabled.

HasClearButton is Boolean. 1 means that a

✕

button will be displayed; 0 (the default) means that the button will not be shown. It may only be specified when the object is created. If you subsequently attempt to change the value of HasClearButton, the operation will fail with `NONCE ERROR`.

HasClearButton is only effective for Edit objects with Style Single; it is silently ignored for other Styles of Edit objects.



6.286 HasEdit Property

Applies To: [Section 6.51](#)

Description

Specifies whether or not a [Section 6.51](#) has an edit field.

HasEdit is a single number with the value 0 (the default) or 1. If HasEdit is 1, the user may type in the name of a folder or other resource that is the target of the [Section 6.51](#). If HasEdit is 0, the user must browse to it.

6.287 HasHelp Property

Applies To: [Section 6.462](#), [Section 6.463](#)

Description

The HasHelp property is a Boolean value. For a [Section 6.463](#), it determines whether or not the [Section 6.463](#) has a Help button. for a [Section 6.462](#), HasHelp determines whether or not the Help button is active when the [Section 6.462](#) is the current page. If the HasHelp property of a [Section 6.462](#) is 0, the Help button on the parent [Section 6.463](#) will be temporarily disabled when that [Section 6.462](#) is displayed.

6.288 HasLines Property

Applies To: [Section 6.629](#)

Description

The HasLines property specifies whether or not tree lines are drawn in a [Section 6.629](#) object. It is a single integer with the value 0, 1 or 2:

0	No tree lines
1	Tree lines are drawn at all levels except the top level
2	Tree lines are drawn at all levels

The user can cause a parent item to expand or collapse by clicking on its corresponding tree line.

6.289 HasTicks Property

Applies To: [Section 6.625](#)

Description

The HasTicks property specifies whether or not tick marks are drawn in a [Section 6.625](#) object. It is Boolean value with a default value of 0.

The position of the tick marks in the [Section 6.625](#) is determined by the [Section 6.611](#) property.

6.290 HasToday Property

Applies To: [Section 6.58](#), [Section 6.143](#)

Description

The HasToday property specifies whether or not the [Section 6.620](#) date is displayed in the bottom left corner of a [Section 6.58](#) object or in the drop-down calendar of a [Section 6.143](#).

HasToday is a single number with the value 0 (the date is *not* shown) or 1 (the date is shown); the default is 1.

See also [Section 6.95](#) property.

6.291 Header Property

Applies To: [Section 6.348](#)

Description

The Header property is Boolean and specifies whether or not a [Section 6.348](#) object displays column titles. Its default value is 1. Header applies only if the [Section 6.644](#) property is 'Report'. The column titles are defined by the ColTitles property and their alignment by the [Section 6.114](#) property.

Note that Header may only be set by `WC` and may not subsequently be changed.

6.292 HeaderImageIndex Property

Applies To: [Section 6.348](#)

Description

The HeaderImageIndex property is an integer vector that specifies the images to be displayed alongside each column heading in a [Section 6.348](#) object in Report View. Each positive element of HeaderImageIndex specifies an index into the [Section 6.316](#) object specified by the [Section 6.293](#) property. The special values -1 and -2 specify the standard *Sort Up* and *Sort Down* images respectively.

6.293 HeaderImageList Property

Applies To: [Section 6.348](#)

Description

The HeaderImageList property specifies the name of or ref to an [Section 6.316](#) object that contains images to be displayed alongside each column heading in a [Section 6.348](#) object in Report View.

6.294 Help Event 400

Applies To

Description

If enabled, this event is reported when the user clicks on an object which has a callback defined for this event, the user having previously clicked on the Question (?) button in the title bar of the parent Form. The presence of the Question (?) button is determined by the value of the [Section 6.295](#) property.

The event message reported as the result of `OnDQ`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' Help or 400
[3]	Y-coordinate	Number
[4]	X-coordinate	Number

The y and x-coordinates refer to the position of the mouse pointer in the object which was clicked on, and are reported in the coordinate system of that object.

6.295 HelpButton Property

Applies To: [Section 6.227](#), [Section 6.463](#), [Section 6.578](#)

Description

This is a Boolean property that specifies whether or not a Question (?) button appears in the title bar of a Form or SubForm. However, this does not apply if the Form has a maximise or minimise button which both take precedence. The user may obtain help by clicking on the Question (?) button and then on a control in the Form. It is up to you to provide the help by responding to the [Section 6.294](#) event on the control. The default value of HelpButton is 0.

6.296 HelpFile Property

Applies To: [Section 6.7](#), [Section 6.412](#), [Section 6.414](#)

Description

The HelpFile property is a read-only property that reports the pathname of a Windows help file associated with a particular COM object.

For an [Section 6.412](#) or [Section 6.414](#) object, the HelpFile property is read-only.

6.297 HideComment Event 224

Applies To: [Section 6.263](#)

Description

If enabled, a HideComment event is generated just before a comment window is hidden as a result of the user moving the mouse-pointer away from a commented cell.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'HideComment' or 224
[3]	Row	integer
[4]	Column	integer

You may prevent the comment from being hidden by returning 0 as the result of a callback function.

Note that if the comment window relates to a row or column *title*, the value reported in element [3] or [4] of the event message is `1`.

Invoked as a method, HideComment is used to hide a comment that has previously been displayed by ShowComment. For example, the following expression hides the comment associated with the cell at row 2, column 1.

```
F.G.HideComment 2 1
```

If HideComment is called with an argument of `0`, all comments are hidden.

6.298 HighlightHeaders Property

Applies To: [Section 6.263](#)

Description

The HighlightHeaders property is a Boolean value (default 1) that specifies whether or not the appropriate row and column titles in a [Section 6.263](#) are highlighted corresponding to the currently selected block of cells.

6.299 Hint Property

Applies To

Description

The Hint property is a character vector that specifies a *help* message that is to be displayed when the user positions the mouse pointer over the object. The Hint is displayed in the object specified by its [Section 6.300](#) property. A [Section 6.575](#) is often used for this purpose.

6.300 HintObj Property

Applies To

Description

The HintObj property is a character vector or ref that specifies the name of, or a ref to, an object in which the "help" message defined by the [Section 6.299](#) property is to be displayed. This message is displayed when the user positions the mouse pointer over the object. The [Section 6.299](#) is displayed by automatically setting the [Section 6.66](#) or [Section 6.606](#) property of the object named by HintObj.

The following types of object can therefore be used to display Hints: [Section 6.55](#), [Section 6.185](#), [Section 6.121](#), [Section 6.279](#), [Section 6.227](#), [Section 6.341](#), [Section 6.382](#), [Section 6.384](#), [Section 6.575](#), [Section 6.578](#) and [Section 6.605](#). For a [Section 6.575](#) that has both [Section 6.66](#) and [Section 6.606](#) properties, the text property is used for displaying hints.

When the user moves the mouse pointer away from the object, the [Section 6.66](#) or [Section 6.606](#) property of the object specified by HintObj is reset to an empty vector.

Note that if HintObj is empty, its value is inherited from its parent. Thus setting HintObj on a [Section 6.227](#) defines the default location for displaying [Section 6.299](#)s for all the controls in that [Section 6.227](#). Setting HintObj on [Section 6.489](#) defines the default location for hints for the entire application.

6.301 HotSpot Property

Applies To: [Section 6.136](#)

Description

This property specifies the point within a [Section 6.136](#) object that registers the cursor's position over another object. The mouse position, which is reported by various events, is actually the position of the cursor's HotSpot over the object in question.

HotSpot is a 2-element numeric vector that specifies the y-position and x-position of the hotspot within the cursor. A value of (0 0) specifies the top-left corner of the cursor; (31 31) specifies the bottom right corner of the cursor. The default value of HotSpot is (16 16).

6.302 HotTrack Property

Applies To: [Section 6.585](#)

Description

The HotTrack property specifies whether or not the tabs or buttons in a [Section 6.585](#) object (which are represented by [Section 6.584](#) objects), are automatically highlighted by the mouse pointer.

HotTrack is a single number with the value 0 (no highlighting) or 1. The default is 0.

If HotTrack is 1 and the Style property of the [Section 6.585](#) is 'Tabs' or 'Buttons', the text defined by the Caption property of the [Section 6.584](#) is highlighted when the mouse pointer is placed over the tab or button. If Style is 'FlatButtons', the button is highlighted by being *raised*.

The value of HotTrack is effective only when the object is created with `WC`.

6.303 HScroll Property

Applies To

Description

For most objects to which it applies, this property determines whether or not a horizontal scrollbar is provided.

HScroll may only be set when the object is created with `WC` and may not subsequently be changed with `WS` or assignment.

When applied to a [Section 6.121](#), or to an [Section 6.185](#) object with [Section 6.577](#) 'Single' (that is, a single-line edit field), the value 0 inhibits scrolling, and prevents the user from entering more data when the field is full. If instead it has the

value -2 , the field is scrollable, and the length of data that may be entered is not limited by the length of the field.

When applied to an [Section 6.185](#) object with [Section 6.577](#) 'Multi' (that is, a multi-line text box), the value 0 inhibits scrolling, and causes individual lines to be "word-wrapped". The values -2 and -1 enable sideways scrolling, and permit individual lines to exceed the width of the object. The value -1 means that a horizontal scrollbar is provided.

For a [Section 6.508](#) object, the scrollbar is horizontal if HScroll is -1 and vertical if HScroll is 0. For a [Section 6.227](#), a horizontal scrollbar is provided if HScroll is set to -1 . The default value is 0 (no scrollbar).

For a [Section 6.574](#), [Section 6.582](#) or [Section 6.621](#) with [Section 6.13](#) set to Top or Bottom, HScroll determines whether or not a horizontal scrollbar is provided and how the object positions its children. If HScroll is 0 (the default) the object organises its children in multiple rows and does not provide a scrollbar. If HScroll is -1 or -2 , the object organises its children in a single row and provides a mini scrollbar to allow those positioned beyond the right edge of the object to be scrolled into view. If HScroll is -1 , the scrollbar is always shown. If HScroll is -2 , it is only shown when needed.

For a [Section 6.263](#), HScroll may be 0 (no horizontal scrollbar), -1 (scrollbar is displayed when required), -2 (same as -1) or -3 (scrollbar is always displayed).

6.304 HScroll Event 39

Applies To: [Section 6.227](#), [Section 6.578](#)

Description

If enabled, this event is generated when the user attempts to move the thumb in a horizontal scrollbar in a [Section 6.227](#) or [Section 6.578](#). This event occurs only in a [Section 6.227](#) whose [Section 6.303](#) property is set to -1 and is distinct from the Scroll event that is generated by a [Section 6.508](#) object. The event may be generated in one of three ways:

- dragging the thumb.
- clicking in one of the "arrow" buttons situated at the ends of the scrollbar. This is termed a small change, the size of which is defined by [Section 6.576](#)[3].
- clicking in the body of the scrollbar. This is termed a large change, the size of which is defined by [Section 6.576](#)[4].

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 4-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'HScroll' or 39
[3]	Scroll Type	numeric
[4]	Position	numeric

The value of Scroll Type is 0 (drag), 1 or $\bar{1}$ (small change) or 2 or $\bar{2}$ (large change). The sign indicates the direction.

The value of Position is the new (requested) position of the thumb. Notice however, that the event is generated **before** the thumb is actually moved. If your callback function returns a scalar 0, the position of the thumb will remain unaltered.

6.305 HThumbDrag Event 442

Applies To: [Section 6.227](#), [Section 6.578](#)

Description

If enabled, this event is generated when the user attempts to drag the thumb in a horizontal scrollbar in a [Section 6.227](#) or [Section 6.578](#). This event occurs only in a [Section 6.227](#) or [Section 6.578](#) whose [Section 6.303](#) property is set to $\bar{1}$ and is distinct from the Scroll event that is generated by a [Section 6.508](#) object.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'HThumbDrag' or 442
[3]	Position	numeric

The value of Position is the new (requested) position of the Thumb. Setting the action code of this event to $\bar{1}$, or returning a 0 from a callback function attached to it, has no effect.

6.306 HTML Property

Applies To: [Section 6.307](#)

Description

The HTML property is a character vector that specifies the content to be rendered by the [Section 6.307](#) object. Dyalog does not perform any pre-processing of the text nor does it verify that it is properly formed HTML using single-byte character data, including any necessary escaping and encoding.

See also: [Section 6.638](#).

Note

Typically, you will need to UTF-8 encode any text outside the Unicode range 0-127.

6.307 HTMLRenderer Object

Purpose: The HTMLRenderer Object is a cross-platform mechanism for producing Graphical User Interfaces (GUI), based on HyperText Markup Language (HTML).

Parents ActiveXControl Form Group PropertyPage Root SubForm

Children Timer

Properties Type HTML Posn Size URL Coord Border Active Visible Event Sizeable Moveable SysMenu MaxButton MinButton IconObj Data Attach Translate KeepOnClose AsChild InterceptedURLs CEFVersion Caption AllowContextMenu IsLoading MethodList ChildList EventList PropList

Methods Detach SetZoomLevel GetZoomLevel ExecuteJavaScript WebSocketClose PrintToPDF WebSocketSend ShowDevTools Wait

Events Close Create LoadEnd HTTPRequest WebSocketUpgrade WebSocketReceive WebSocketError DoPopup SelectCertificate

Description

The HTMLRenderer object renders HTML in a window on the screen. It may appear as a top-level window, similar to a Form, or be displayed within another GUI object according to the value of the Boolean [Section 6.31](#) property which must be specified when the HTMLRenderer is created. Several HTMLRenderer objects may co-exist in the Dyalog application.

The HTMLRenderer is supported on the following platforms:

- Microsoft Windows
- macOS (ARM-based only)
- Linux (x86_64-based only)

It is not supported on the Raspberry Pi.

The HTMLRenderer is implemented using the [Chromium Embedded Framework \(CEF\)](#). Note that if the **ENABLE_CEF parameter** is set to 0 (its default value is 1) the CEF is disabled and an attempt to create an HTMLRenderer object will fail with an error message.

The HTMLRenderer object can be considered as two components; a client implemented using CEF and an internal server which implements an interface from the APL workspace to the client. The client may communicate with both the internal server and external servers on the web. Thus it can combine and display information from external and internal feeds.

Internal and external communications are distinguished primarily by the [Section 6.326](#) property which specifies which requests from the HTMLRenderer client component are to be handled by the internal server component and which are to be serviced by the internet. The default value of [Section 6.326](#) has been chosen so that in most cases it can be ignored.

The HTMLRenderer object supports both the HTTP protocol and the WebSocket (WS) protocol.

Using the HTTP protocol, the client requests a resource, such as a style-sheet, an image, or a complete web page, which the server then delivers. All communication is initiated by the client and involves the creating, use, and closing of a TCP/IP socket.

Using WS protocol, the client asks the server for a permanent communications channel (this is done by *upgrading* the TCP/IP socket to a WebSocket) which may subsequently be used for messages initiated by either the client or the server.

The internal server component of the HTMLRenderer is implemented by functions in the workspace.

HTTP protocol communications are handled by callback functions on the [Section 6.308](#) event.

WS protocol communications are handled:

- by the [Section 6.655](#) event, which reports the initial connection and the WebSocket ID,
- by a callback on the [Section 6.653](#) event
- and by calling the [Section 6.654](#) method.

The HTMLRenderer may be initialised by setting its [Section 6.306](#) property to a character vector representing a base HTML document. This will typically contain references to other documents, such as JavaScript and CSS files which contain code that can influence the way the base HTML is rendered, image files in a variety of formats, and of course hyperlinks to other pages.

If the HTML contains references to other documents, the CEF will retrieve each one by making an HTTP request. Requests with URLs that match a triggering pattern in the [Section 6.326](#) property will generate an [Section 6.308](#) event on the HTMLRenderer, which can be directed to a callback function.

Requests with URLs that do not match a pattern in [Section 6.326](#), or that match a pattern with a 0 in the second column, will be handled by the CEF.

Requests handled by the CEF push the request out to the network to be serviced by an external web server and require that the system has an active internet connection.

An alternative is to initialise the HTMLRenderer by setting its [Section 6.638](#) property. This is typically used to display external content , rather than content delivered from the workspace.

If neither [Section 6.306](#) nor [Section 6.638](#) is set when the HTMLRenderer is created, it will generate an [Section 6.308](#) event with a requested url of `http://dialog_root`.

When the HTMLRenderer is displayed in its own window, the window caption is set by an assignment to its [Section 6.66](#) property. The window caption may subsequently change when content is displayed (typically by the title tag in the html). The [Section 6.66](#) property reports the current window caption.

EXAMPLE

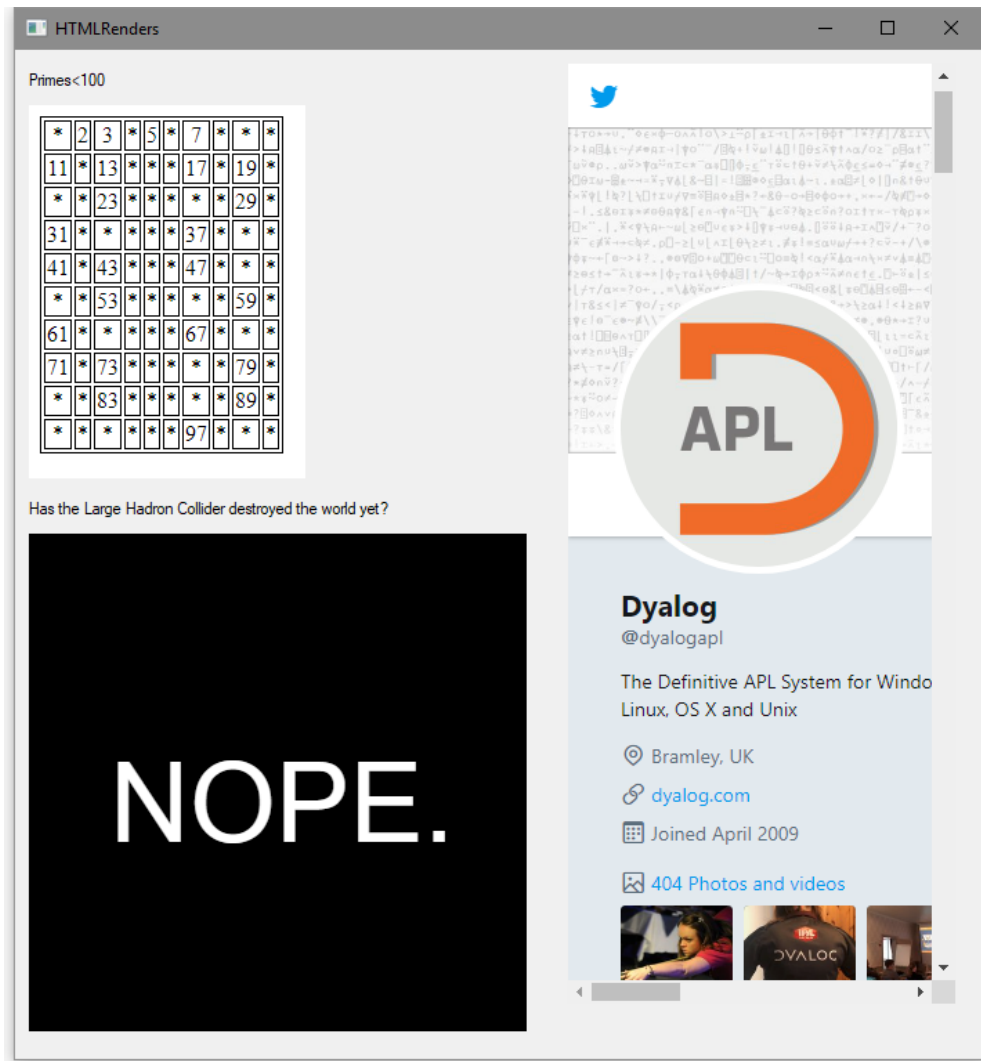
```

▼ Example;Q;U;tw
[1]   'f'␣WC'Form' 'HTMLRender'
[2]   f.(Coord Size)←'Pixel'(730 700)
[3]   'pco'␣CY'dfns'
[4]   'f.l1'␣WC'Label' 'Primes<100'(10 10)
[5]   'f.p'␣WC'HTMLRenderer'('AsChild' 1)('Posn' 40 10)(270 200)
[6]   f.p.HTML←HTMLTable('*'␣@(0␣pco)10 10␣1100)
[7]   Q←'Has the Large Hadron Collider destroyed the world yet?'
[8]   'f.l2'␣WC'Label'Q(320 10)
[9]   'f.q'␣WC'HTMLRenderer'('ASChild' 1)
[10]  f.q.(Posn Size)←(350 10)(360 360)
[11]  U←'https://hashthelargehadroncolliderdestroyedtheworldyet.com'
[12]  f.q.URL←U
[13]  tw←'<a class="twtimeline"'
[14]  tw,←'href="https://twitter.com/dyalogapl">'
[15]  tw,←'Tweets by dyalogapl</a>'
[16]  tw,←'<script async src="https://platform.twitter.com/widgets.js"'
[17]  tw,←'charset="utf-8"></script>'
[18]  'f.t'␣WC'HTMLRenderer'('AsChild' 1)
[19]  f.t.(Posn Size)←(10 400)(680 280)
[20]  f.t.HTML←tw
▼

```

```
enc←{'<',α,'>',(∈ℱω),'</',((~v\' '=α)/α),'>'}
```

```
HTMLTable←{'table border="1"'enc∈(<'tr')enc◦∈"↓(<'td')enc"ω}
```



For more information on the HTMLRenderer, see the [HTMLRenderer User Guide](#).

6.308 HTTPRequest Event 840

Applies To: [Section 6.307](#)

Description

An HTTPRequest event is raised whenever the [Section 6.307](#) requests a url from the workspace. See [Section 6.326](#) property. The request could be generated by a form submission, clicking on a hyperlink, an AJAX request or a link to a resource like a style

sheet, image or JavaScript file. An HTTPRequest event is also raised when the HTMLRenderer is initialised and both [Section 6.306](#) and [Section 6.638](#) are empty.

The callback function must "fill in the blanks" in the event message and return the modified event message as its result.

The event message reported as the result of `DDQ`, or supplied as the right argument to your callback function, is an 11-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'HTTPRequest' or 840
[3]		'ProcessRequest' (unused)
[4]	Handle	Initially 0, must be set to 1.
[5]	Status	Integer HTTP status code (initially 0).
[6]	Message	Character vector containing the HTTP status message (initially empty).
[7]	MIME	Character vector containing the MIME type (initially empty). See below.
[8]	URL	Character vector containing the requested URL.
[9]	Headers	Character vector containing the HTTP Request headers (initially empty).
[10]	Body	Character vector containing the HTTP Request body (initially empty).
[11]	Method	Character vector containing the HTTP method, for example, 'GET' or 'POST'.

To process the request, the callback function should return the message with only the following items changed. Note that only elements [4 5 6 10] are always required, and it is important to set element [4] to 1.

[4]	Handle	1
[5]	Status	Success is indicated by 200.
[6]	Message	Success is indicated by 'OK' .
[7]	MIME	Defaults to 'text/html' and need be specified only if the response (Body) is not a character vecttor containing HTML.
[9]	Headers	Not normally required.
[10]	Body	This may be a character vector or an integer numeric vector with values in the ranges <code>[-128-127]</code> or <code>[0-255]</code> . Note that a character vector will be prepended with a UTF-8 byte order mark (BOM). A numeric vector requires that [7] is an appropriate MIME type (for example, 'image/png')

MIME types include:

- text/html
- text/css
- text/plain
- text/csv
- application/javascript
- application/xml
- application/json

For a complete list of media/MIME types, see:

<https://www.iana.org/assignments/media-types/media-types.xhtml>:

EXAMPLE

```

<!DOCTYPE html>
<html>
<head>
<Title>HTTPRequest Example</Title>
</head>
<body><h1>Simple Form</h1>

<form action="Hello">
  First name:<br>
  <input type="text" name="firstname" value="Mickey">
  <br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse">
  <br><br>
  <input type="submit" value="Submit">
</form>

<p>When you click the "Submit" button, the HTMLRenderer
will fire an HTTPRequest event.</p>

</body>
</html>

```

```

▼ msg←SayHello msg:url;names;first;last;response
[1]   url←8>msg
[2]   'Requested URL is: ',url
[3]   names←(url←'?&')<url
[4]   first last←{(ω1'=')&ω}names
[5]   response←'<!DOCTYPE html><html><head>'
[6]   response,←'<Title>Hi Folks</Title>'
[7]   response,←'</head><body>'
[8]   response,←'<h1 align="center">Hello '
[9]   response,←first,' ',last,'</h1>'
[10]  response,←'</body></html>'
[11]  msg[4 5 6 10]←1 200 'OK' response
▼

```

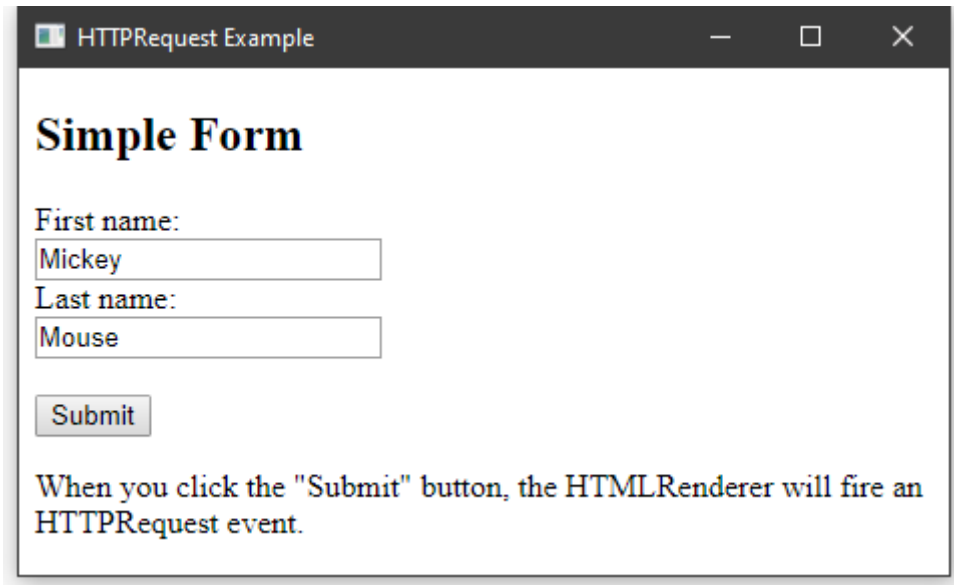
```

▼ run
[1]   'hr'␣WC'HTMLRenderer'
[2]   hr.Size←25 25
[3]   hr.HTML←html
[4]   hr.onHTTPRequest←'SayHello'
▼

```



```
run  
Requested URL is: http://dyalog_root/Hello?firstname=Mickey&lastname=Mouse
```

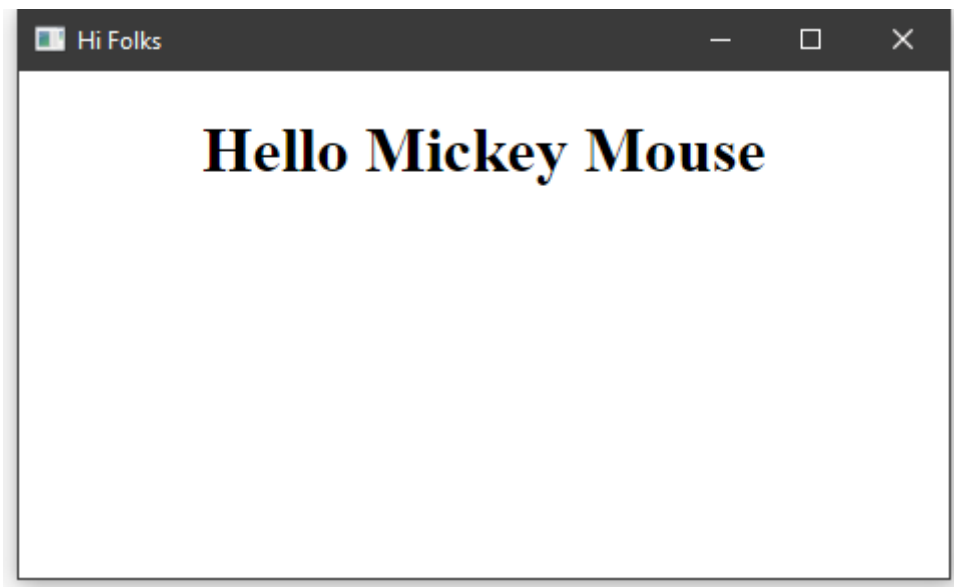


Simple Form

First name:

Last name:

When you click the "Submit" button, the HTMLRenderer will fire an HTTPRequest event.



6.309 Icon Object

Purpose: This object defines an icon.

Parents ActiveXControl CoolBand Form Grid ImageList ListView OLEServer Printer
 ProgressBar PropertyPage PropertySheet RichEdit Root StatusBar SubForm SysTrayItem
 TCPSocket ToolBar ToolControl TrackBar TreeView UpDown

Children Timer

Properties Type File Bits CMap Mask Style KeepBits Size Event Data Handle Accelerator
 KeepOnClose CBits MethodList ChildList EventList PropList

Methods Detach FileRead FileWrite

Events Close Create Select

Description

The [Section 6.207](#) property specifies the name of an icon file (.ICO, .GIF or .PNG), or the name of a DLL or EXE file and the identity of the icon within it.

The Style property identifies the size of the icon and must be 'Large' or 'Small'. The former specifies a 32x32 icon and is the default; the latter specifies a 16x16 icon. The size of the icon is not embedded within the icon data, so it is **essential** to specify Style correctly. Note that a single file may contain both sizes of an icon. Style is only relevant when loading an Icon from file.

If the value of the [Section 6.207](#) property is set by *Dyalog APL Language: Ws*, no immediate action is taken, but the corresponding file may subsequently be read or written using the [Section 6.212](#) or [Section 6.213](#) methods.

16-bit icons contain fewer than 256 colours and each pixel is either transparent or opaque. The images in such Icons are represented by the [Section 6.48](#), [Section 6.366](#) and [Section 6.104](#) properties.

32-bit icons are 24-bit images with an 8-bit alpha channel which specifies the degree of transparency of each pixel. The pixels in these Icons are represented by the [Section 6.68](#) property.

[Section 6.68](#) is a rank-2 numeric array whose dimensions represent the rows and columns of pixels in the Icon. The values in [Section 6.68](#) represent the colour and of each pixel and also its transparency.

[Section 6.48](#) is an integer matrix whose elements define the colours of each pixel in the icon in terms of their (0-origin) indices into [Section 6.104](#). When the icon is displayed on the screen, the way in which these colours combine with those currently displayed on the screen (the background) is specified by [Section 6.366](#). This is a Boolean matrix of

the same size as [Section 6.48](#). The following table shows how the colour of each resulting pixel is determined.

Bits	Colour	0	Colour
Mask	0	1	1
Pixel	Colour	Background	New Colour

If an element of [Section 6.366](#) is 0, the corresponding element of [Section 6.48](#) defines the colour of the resulting pixel that is displayed on the screen. If an element of [Section 6.366](#) is 1, the resulting pixel that is displayed on the screen is either the current background colour or is a new colour chosen by MS-Windows to be visible against the background. A non rectangular icon is obtained by setting those elements of [Section 6.48](#) and [Section 6.366](#) that you want to exclude from the shape to be 0 and 1 respectively.

The size of [Section 6.48](#) is restricted by the capabilities of the current display driver. [Section 6.366](#) must have the same shape as [Section 6.48](#).

An Icon is **used** by setting the [Section 6.310](#) property or [Section 6.447](#) property of another object to its name or ref.

6.310 IconObj Property

Applies To

Description

This property is used to specify a *large* and *small* icon for a [Section 6.227](#) or [Section 6.578](#), or for the [Section 6.489](#) object which represents your application as a whole. Its value is either a single ref or character scalar or vector containing the name of, or ref to, an Icon object, or a 2-element vector of character vectors or refs that specifies 2 [Section 6.309](#) objects.

If empty (the default value), the standard "Dyalog APL GUI" icon is used.

The large and small icons are supplied to the Operating System which uses them as and when is appropriate. Normally, the large icon is of size 32x32 and the small icon is 16x16. If you specify an icon of a different size, the Operating System will scale it as appropriate.

The icon associated with a [Section 6.227](#) is displayed when the [Section 6.227](#) is minimised. The icon associated with ' . ' is displayed when the user presses Alt+Tab to

toggle between applications. In both cases, the text shown underneath or alongside the icon is defined by the [Section 6.66](#) property.

6.311 Idle Event 130

Applies To: [Section 6.489](#)

Description

If enabled, this event is generated whenever APL looks to see if there is an event on the queue and finds it empty. Its purpose is to allow an application to perform some background processing when the user is not doing anything. It is unwise to use this event directly from the Session as it will occur repeatedly and may lock you out.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function is a 2-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'Idle' or 130

6.312 IDNToDate Method 263

Applies To: [Section 6.58](#), [Section 6.143](#), [Section 6.489](#)

Description

This method is used to convert a date from an [Section 8.5](#) into $\square$ TS format (year, month, day). The corresponding day of the week is also obtained.

The argument to IDNToDate is a single item as follows:

[1]	IDN	Integer

The result is a 4-element integer vector containing the year, month, day, and weekday corresponding to the IDN that was specified.

The value of the 4th element, weekday, is an integer in the range 0-6 that specifies on which day of the week the specified date falls (0=Monday).

EXAMPLE

```
F.C.IDNToDate 36048
1998 9 11 4
```

6.313 Image Object

Purpose: Positions bitmaps and icons within an object.

Parents ActiveXControl Bitmap Form Grid Group Metafile Printer PropertyPage Static StatusBar SubForm ToolBar ToolControl

Children Timer

Properties Type Points Coord Visible Event Draggable Picture OnTop AutoConf Data EdgeStyle Size Accelerator AcceptFiles KeepOnClose MethodList ChildList EventList PropList

Methods Detach

Events Close Create DragDrop MouseDown MouseUp MouseMove MouseDbClick Help Select

Description

The [Section 6.449](#) property specifies the co-ordinates of one or more points at which the specified graphical objects are to be drawn.

The [Section 6.447](#) property specifies the name(s) of [Section 6.47](#), [Section 6.309](#) or [Section 6.385](#) object(s) that are to be drawn. It may be a simple character vector or a vector of vectors.

To draw a single graphic picture, the [Section 6.447](#) property is a simple character vector specifying the name of a [Section 6.47](#), [Section 6.309](#) or [Section 6.385](#) object. [Section 6.449](#) is either a 2-element vector or a 1-row, 2-column matrix whose elements specify the y-coordinate and x-coordinate respectively at which the object is to be drawn.

To draw the same picture at several different positions, the [Section 6.447](#) property is a simple character vector specifying the name of the [Section 6.47](#), [Section 6.309](#) or [Section 6.385](#) object. [Section 6.449](#) is either a 2-column matrix of y-coordinates and x-coordinates, or a nested vector whose first element contains the y-coordinates and whose second element contains the x-coordinates.

To draw several different pictures, the [Section 6.447](#) property is a vector of character vectors specifying the names of several [Section 6.47](#), [Section 6.309](#) and/or [Section 6.385](#) objects. [Section 6.449](#) is a 2-column matrix or 2-element nested vector as described above.

Setting the [Section 6.184](#) property causes the picture to be surrounded by the appropriate border. For example, setting [Section 6.184](#) to 'Plinth' produces a button-like appearance.

Setting the [Section 6.556](#) property causes the picture to be scaled to fit within the specified rectangle. It is only necessary to specify [Section 6.556](#) when an Image is used to draw a [Section 6.385](#) object. For a [Section 6.47](#) or [Section 6.309](#), [Section 6.556](#) defaults to the size of the object being drawn.

The [Section 6.173](#) property specifies whether or not the Image can be dragged and dropped using the mouse.

EXAMPLES

First make a [Section 6.227](#)

```
'F' WC 'Form'
```

Then make two [Section 6.47s](#) :

```
'YES' WC 'Bitmap' 'C:\WDYALOG\WS\YES'  
'NO' WC 'Bitmap' 'C:\WDYALOG\WS\NO'
```

Display the "YES" [Section 6.47](#) at (20,10)

```
'F.I' WC 'Image' (20 10)('Picture' 'YES')
```

Display the "YES" [Section 6.47](#) at (20,10) and (20,50)

```
'F.I' WC 'Image' (20(10 50))('Picture' 'YES')
```

Display the "YES" [Section 6.47](#) at (20,10) and the "NO" [Section 6.47](#) at (20,50)

```
'F.I' WC 'Image' (20(10 50))('Picture' 'YES' 'NO')
```

6.314 ImageCount Property

Applies To: [Section 6.316](#)

Description

The ImageCount property is a read-only property that reports the number of images in an [Section 6.316](#) object. It is an integer scalar.

6.315 ImageIndex Property

Applies To

Description

For a [Section 6.122](#), [Section 6.348](#) or [Section 6.629](#), the ImageIndex property maps bitmapped images in an [Section 6.316](#) to items. ImageIndex is an integer vector whose length is the same as the number of items in the object. See also [Section 6.518](#)

For a [Section 6.126](#), [Section 6.384](#), [Section 6.584](#) or [Section 6.623](#), ImageIndex specifies the picture to be displayed in the object. In these cases, ImageIndex is a single integer value.

ImageIndex is $\square$ IO dependent.

6.316 ImageList Object

Purpose: The ImageList object represents a set of bitmapped images.

Parents ActiveXControl ButtonEdit CoolBand CoolBar Form Group ListView OLEServer PropertyPage Root SubForm TabControl TCPSocket ToolBar ToolControl TreeView

Children Bitmap Cursor Icon Timer

Properties Type Size Event Coord Data Handle Translate ImageCount Masked MapCols KeepOnClose MethodList ChildList EventList PropList

Methods Detach

Events Close Create

Description

An ImageList object represents an array of bitmapped images which are used to depict items in a [Section 6.348](#) or [Section 6.629](#) object, or the images in a [Section 6.127](#), [Section 6.382](#), [Section 6.585](#) or [Section 6.624](#).

Making an ImageList is a 2-step process. First, you create an (empty) ImageList specifying its [Section 6.556](#) and [Section 6.368](#) properties. The former establishes the

size of each of the bitmapped images in the array. The [Section 6.368](#) property specifies whether the ImageList is to contain opaque or transparent images. Note that these properties must be established when the ImageList is created by `WC` and may not subsequently be changed using `WS`.

Next, you create a series of [Section 6.47](#) or [Section 6.309](#) objects as *children* of the ImageList. As you make each one, APL adds the corresponding image (or images) to the ImageList object. If the size of each of the [Section 6.47](#) or [Section 6.309](#) objects is equal to the [Section 6.556](#) of the ImageList itself, each child object corresponds to an image in the ImageList. However, if you add an object whose *width* is an exact multiple of the *width* of the ImageList, a corresponding number of images will be added.

For example, if the [Section 6.556](#) of the ImageList is 16x16 (the default) and you create a child [Section 6.47](#) of size 16x48, three images (each of size 16x16) will be added to the ImageList. This is more efficient than building the images one-by-one. In other circumstances (where the size of the [Section 6.47](#) or [Section 6.309](#) is not equal to [Section 6.556](#) of ImageList), the [Section 6.47](#) or [Section 6.309](#) will be scaled to fit.

Note that when making Bitmaps or Icons as children of an ImageList, it is not necessary to *name* them because they are subsequently referenced only via the [Section 6.315](#) and [Section 6.518](#) properties and not by name. The number of images in an ImageList is given by the read-only property, ImageCount.

The [Section 6.364](#) property, which must be specified at the time you create the object, specifies whether or not bitmap colours are remapped to reflect the user's colour preferences.

An ImageList is *associated* with a [Section 6.348](#) or [Section 6.629](#) object by the [Section 6.317](#) property. Each item in the [Section 6.348](#) or [Section 6.629](#) is then allocated a specific image in the ImageList by the [Section 6.315](#) and [Section 6.518](#) properties.

6.317 ImageListObj Property

Applies To

Description

The ImageListObj property is a simple character vector or a ref, or a vector of character vectors or refs that specifies [Section 6.316](#) objects that are associated with an object.

For a [Section 6.122](#) or [Section 6.629](#) object, the ImageListObj property specifies the name of, or ref to, a single [Section 6.316](#) object that contains a set of images to be displayed alongside its Items. The image(s) displayed by a particular item in its normal

(unselected) and selected states are specified by the corresponding element of the [Section 6.315](#) and [Section 6.518](#) properties respectively.

For [Section 6.127](#), [Section 6.382](#), and [Section 6.585](#) objects, the `ImageListObj` property specifies the name of, or ref to, a single [Section 6.316](#) object that contains a set of images for its [Section 6.126](#), [Section 6.384](#), and [Section 6.584](#) children respectively.

For a [Section 6.624](#), `ImageListObj` may specify up to three [Section 6.316](#) objects that correspond to the three different states, normal, highlighted (hot) and inactive, of its [Section 6.623](#) children.

In all these cases, individual images are mapped to the child objects by their [Section 6.315](#) property.

For a [Section 6.348](#) either one or two [Section 6.316](#) objects may be specified. The first [Section 6.316](#) contains the *large icon* set of images. the second contains the *small icon* set. The set that is used is determined by the value of the [Section 6.644](#) property. The mapping between the set of images in the [Section 6.316](#) and items in the object is determined by the [Section 6.315](#) property.

6.318 Indents Property

Applies To: [Section 6.122](#)

Description

Specifies the amount by which items in a `ComboEx` object are indented.

Indents may be an integer scalar or a vector with the same number of elements as there are items in the `ComboEx`. Its default value is 0.

The unit of indenting is 10 pixels. For example, if there are 3 items and Indents is (0 1 2), the items will be indented by 0, 10 and 20 pixels respectively.

6.319 Index Property

Applies To

Description

For a [Section 6.346](#) and a [Section 6.121](#) with [Section 6.577](#) 'Simple', this property specifies the position of the data in the list box as a positive integer value. If Index has the value "n", it means that the "nth" item in [Section 6.334](#) is displayed on the top line

in the list box. The value of Index is dependent upon the value of `□IO`. Note that Index for a [Section 6.121](#) or [Section 6.346](#) cannot be set using *Dyalog APL Language: Wc*. The value of Index in a [Section 6.121](#) with a drop-down list box ([Section 6.577](#) 'Drop' or 'DropEdit') is always equal to `□IO`.

For a [Section 6.263](#), Index is a 2-element vector that specifies the row and column number of the cell that is currently in the top left corner of the [Section 6.263](#).

For a [Section 6.629](#), Index is a positive integer value that specifies which item appears at the top of the [Section 6.629](#) window.

For a [Section 6.208](#), the Index property determines which of the [Section 6.215](#) is initially selected.

For a [Section 6.126](#), the Index property specifies the position of the [Section 6.126](#) within its parent [Section 6.127](#), relative to the other [Section 6.126](#) in the [Section 6.127](#).

The value of Index is dependent on `□IO`, and its default value is equal to `□IO`.

6.320 IndexChanged Event 210

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported when the value of the Index property of a Grid has changed as a result of user interaction. The event is reported *after* the Grid has been scrolled. You may not modify or nullify the operation with a 0-return callback and you may not call `IndexChanged` as a method or generate this event using `□NQ`. To cause a Grid to scroll, use `□WS` to set its Index property.

The event message reported as the result of `□DQ`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' IndexChanged or 210
[3]	Row	Integer.
[4]	Column	Integer.

6.321 Input Property

Applies To: [Section 6.263](#)

Description

This property specifies objects to be associated with cells in a Grid.

These objects may be of type [Section 6.55](#), [Section 6.108](#), [Section 6.121](#), [Section 6.185](#), [Section 6.341](#), [Section 6.563](#) or [Section 6.625](#). In addition, the Input property may specify instances of [Section 6.412](#) objects (ActiveX controls) and [Section 6.408](#) objects (.NET classes).

The Input property is either a single ref or a simple character scalar or vector, or a vector of character vectors or refs. If it specifies a single object, this will be associated with *all* of the cells in the [Section 6.263](#). If it specifies a set of objects, these are mapped to individual cells through the [Section 6.82](#) property.

When a cell becomes the current cell, its value (defined by the appropriate element of the [Section 6.642](#) property) defines the value of a *corresponding property* of the associated object. The property that corresponds to the value in the cell, depends upon the [Section 6.630](#) of the associated object as shown in the following table:

Associate Object Type	Corresponding Property
<u>Label</u>	<u>Text</u>
<u>Edit</u>	<u>Value</u>
<u>Combo</u>	<u>Text</u>
<u>Button</u> (<u>Style</u> Push)	<u>Caption</u>
<u>Button</u> (<u>Style</u> Radio or Check)	<u>State</u>
<u>ColorButton</u>	<u>CurrentColor</u>
<u>Spinner</u>	<u>Value</u>
<u>TrackBar</u>	<u>Thumb</u>
<u>OCXClass</u>	Specified by <u>InputProperties</u>
<u>NetType</u>	Specified by <u>InputProperties</u>

In effect, the user inputs a new value into the current cell by changing the corresponding property of its associated object. An associated object may be a *fixed*

object that is *external* to the [Section 6.263](#) or a *floating* object that moves automatically from cell to cell. The latter is achieved by creating the associated object as a *child* of the [Section 6.263](#).

If the associated object is an [Section 6.185](#) or [Section 6.121](#), the user may change the [Section 6.606](#) property of the object by typing, or, in the case of a [Section 6.121](#), by selecting an item from a list. The new value of the [Section 6.606](#) property is then used to update the value in the cell (defined by the [Section 6.642](#) property of the [Section 6.263](#)) when the user moves on. If the associated object is a radio button, the value in the cell (0 or 1) is reflected by the [Section 6.571](#) property of the [Section 6.55](#). The user may click the [Section 6.55](#) on and off, changing its [Section 6.571](#) and thereby the corresponding value in the cell.

If the associated object is a [Section 6.108](#), the corresponding elements of the [Section 6.642](#) property contain 3-element integer vectors which specify the RGB colour values.

If the associated object is an instance of an [Section 6.412](#) object (ActiveX control) or a [Section 6.408](#) object (.NET class), the Grid uses the *default* property of the external object if it has one. Alternatively, the [Section 6.324](#) property is used to specify which property (or properties) of the external object are to be mapped to elements of [Section 6.642](#). If more than one property is specified, elements of [Section 6.642](#) are vectors.

If there is no object associated with a cell, or if its associated object is a [Section 6.341](#) or a [Section 6.55](#) with [Section 6.577](#) Push, the cell is protected and may not be changed by the user. When the current cell is thus protected, the corresponding row and column titles are not indented as they are when the cell may be edited.

If the associated object is a numeric [Section 6.185](#) or [Section 6.341](#) ([Section 6.206](#) Numeric, LongNumeric, Currency, Date, LongDate or Time) the contents of the cell are formatted accordingly, even when it is not the current cell. Thus a cell associated with a [Section 6.341](#) with [Section 6.206](#) Date, always displays as a date.

If the associated object is a [Section 6.121](#) or [Section 6.55](#), the appearance of a non-current cell is defined by the corresponding element of the [Section 6.550](#) property.

The following example illustrates the use of different types of object specified by the Input and [Section 6.550](#) properties:

Different Input Objects						
Surname	Job Title		Region		Salary	Permanent
Brown	Manager	↓	South	↓	\$64000.00	☒
Jones	Project Leader	↓	South	↓	\$43250.00	☒
Green	Consultant	↓	South	↓	\$45000.00	☐
Black	Programmer	↓	East	↓	\$30000.00	☒
White	Assistant	↓	Central	↓	\$40000.00	☐

6.322 InputMode Property

Applies To: [Section 6.263](#)

Description

This property determines editing behaviour and the action of the cursor movement keys when the user changes the contents of a Grid using a *floating* [Section 6.185](#) or DropEdit [Section 6.121](#) control.

InputMode is a character vector with one of the following values:

'Scroll'	The cursor keys move around the Grid; the user may switch to <i>InCell</i> mode
'InCell'	The cursor keys move within the Input object; the mode reverts to <i>Scroll</i> when the user selects a new cell
'AlwaysScroll'	The cursor keys move around the Grid; the user may not switch to <i>InCell</i> mode
'AlwaysInCell'	The cursor keys move within the Input object, even when the user moves to a new cell
'AutoEdit'	See below

By default, the input mode is *Scroll*. In this mode, cursor movement keys are actioned by the Grid itself and used to move from cell to cell. The user may switch to *InCell* mode by double-clicking or by pressing the key defined by [Section 6.323](#) (the default is "F2").

In *InCell* mode, all cursor movement keys are actioned by the Input object and typically move the cursor around *within* the Input object and do not switch between cells. When the user switches to a different cell, InputMode reverts to *Scroll* mode

If InputMode is *AlwaysScroll* or *AlwaysInCell*, the user remains permanently in either *Scroll* or *InCell* mode respectively.

If InputMode is 'AutoEdit', the behaviour of a cell that contains a floating Input field is as follows:

When the user enters the cell, the contents are selected (and highlighted). At this stage, the cursor movement keys move to an adjacent cell. If the user presses a (valid) data key, that character replaces the current contents of the cell.

If the user presses F2 (or the key defined by the [Section 6.323](#) property), the data is de-selected and unhighlighted and the cursor is placed at the rightmost end of the data.

In either case, the left and right cursor keys now move the cursor within the current data string, but skip to the adjacent cell from the beginning or end of the data. This behaviour differs from *InCell* mode in which the cursor movement keys *stick* at the end of the data.

6.323 InputModeKey Property

Applies To: [Section 6.263](#)

Description

This property defines the keystroke used to switch from Scroll mode to InCell mode in a [Section 6.263](#). It applies only where the [Section 6.263](#) has a *floating* [Section 6.185](#) control. See the description of the [Section 6.322](#) property for further details.

The InputModeKey property is specified (in the same way as the [Section 6.2](#) property) as a 2-element vector of integer values containing the key number and shift state respectively. Its default is (113 0) which is "F2".

As an example, if you wanted to use Ctrl+Shift+a to switch modes, you would set InputModeKey to (65 3). 65 is the keynumber for "a" and 3 means Shift (1) + Ctrl (2).

6.324 InputProperties Property

When an ActiveX Control or .NET Class is used as a child of the [Section 6.263](#), InputProperties is used to specify how the value in each [Section 6.263](#) cell corresponds to the value of one or more properties of the child object.

For example, suppose there is a third-party ActiveX Control that displays a playing card. The control has two properties named Suit and Value that specify the suit (1=clubs, 2=diamonds, 3=hearts, 4=spades) and card value (1="Ace", 2="2", ...11="Jack",...) respectively. To display these cards in a [Section 6.263](#), the InputProperties property may be set to ('Suit' 'Value') and each element of the [Section 6.642](#) property must be a 2-element integer vector specifying the suit and value of the corresponding card.

```
'CARDS' □WC 'OCXClass' '...'
'F' □WC 'Form'
'F.G' □WC 'Grid'
'F.G.card' □WC 'CARDS'
F.G.Input ← 'F.G.card'
F.G.InputProperties ← 'Suit' 'Value'
F.G.Values ← t4 13
```

If InputProperties is not specified, the *default* property of the ActiveX Control or .NET Class is used.

6.325 InstanceMode Property

Applies To: [Section 6.414](#)

Description

The InstanceMode property specifies how APL attempts to connect the [Section 6.414](#) to an OLE Server.

InstanceMode is a character vector that may be 'ExistingFirst' (the default), 'ExistingOnly' or 'New'. Its value is effective only when the object is created with □WC. Changing InstanceMode with □WS has no effect.

If InstanceMode is 'ExistingFirst', APL attempts first to connect to a running instance of the OLE Server. If there is no running instance, it starts the OLE server to obtain a new object.

If InstanceMode is 'ExistingOnly', APL attempts to connect to a running instance of the object. If there is no running instance, it fails with a DOMAIN ERROR.

Note that in either case, if there is more than one instance running, there is no way to predict to which instance APL will be connected.

If InstanceMode is 'New', APL attempts to start the OLE Server to obtain a new object, whether or not the OLE Server is already running. However, if the OLE Server has registered itself as a single instance object and is already running, APL will be connected to it, and a second instance of the Server will not in fact be started.

6.326 InterceptedURLs Property

Applies To: [Section 6.307](#)

Description

The InterceptedURLs property is a 2-column matrix that specifies whether the [Section 6.307](#) will attempt to satisfy a request for a resource from the workspace or, via the CEF, from the internet. If directed to the workspace, the request will trigger an [Section 6.308](#) event if the protocol is `http`, or a [Section 6.655](#) event if the protocol is `ws`.

The first column is a wild-carded character scalar or vector containing a pattern to match. The second column is numeric indicating whether or not the [Section 6.307](#) should trigger an event as shown in the table below. InterceptedURLs may contain any number of rows.

Value	Meaning
0	Ignore request; pass to CEF for fulfilment
1	Trigger an HTTPRequest or WebSocketUpgrade (automatic) in the workspace
2	Trigger an HTTPRequest or WebSocketUpgrade (manual) in the workspace

If the requested url is a relative rather than an absolute URL, it is prepended by the string `http://dialog_root/`. So, for example, if the [Section 6.306](#) property contains :

```
<link rel="stylesheet" href="style.css">
<script src="app.js"></script>
```

the HTMLRenderer will request `http://dialog_root/style.css` and `http://dialog_root/app.js` respectively.

When the value of InterceptedURLs is its default (`(0 2p'')`) it is treated as if it were set to (`((1 2p'*://dialog_root/*' 1)`). So by default, requests for a relative URL will fire

an event in the workspace while absolute URL will be directed by the CEF to the internet.

Note that if code in the page creates a web socket intended for internal use, with anything other than `dya log_root` as the URL, the URL must match a pattern in `InterceptedURLs` with 1 in the second column. The following example does not require a matching pattern in `InterceptedURLs`.

```
// Create a new WebSocket.
window.socket = new WebSocket('ws://dya log_root/');
```

EXAMPLES

The following will trigger an event for all requested URLs.

```
InterceptedURLs ← 1 2p '*' 1
```

The following will attempt to retrieve from the net URLs containing `'.dya log.com'` and trigger an [Section 6.308](#) event for all other requested URLs.

```
InterceptedURLs ← 2 2p '*.dya log.com*' 0 '*' 1
```

6.327 Interval Property

Applies To: [Section 6.459](#), [Section 6.613](#)

Description

Interval is an integer value specified in milliseconds and has a default of 1000.

For a [Section 6.613](#) object, the Interval property specifies the frequency with which it generates [Section 6.614](#) events. Setting Interval to 0 disables [Section 6.614](#) events.

For a [Section 6.459](#) with [Section 6.461](#) 'Marquee', Interval specifies the frequency with which the animated bar is updated. Setting Interval to -1 stops the animation.

6.328 IsLoading Property

Applies To: [Section 6.307](#)

Description

IsLoading is a Boolean property whose value is 1 if the browser is currently loading, or 0 when the frame content is completely loaded.

6.329 Italic Property

Applies To: [Section 6.222](#)

Description

This property specifies whether or not a font represented by a [Section 6.222](#) object is italicised or not. It is either 0 (normal) or 1 (italic). There is no default; the value of this property reflects the characteristic of the selected font.

6.330 ItemDbClick Event 342

Applies To: [Section 6.348](#), [Section 6.629](#)

Description

If enabled, this event is reported when the user double-clicks a mouse button when the mouse pointer is over an item in a [Section 6.348](#) or [Section 6.629](#) object. This event is reported for information only and may not be controlled in any way using a callback function. Generating the event with `□NQ`, or calling it as a method, has no effect.

The event message reported as the result of `□DQ`, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ItemDbClick' or 342
[3]	Item number	Integer. The index of the item.
[4]	Mouse button	Integer.
[5]	Shift state	Integer. Sum of 1=shift key, 2=Ctrl key, 4=Alt key
[6]	Position	Integer. Indicates the position of the mouse-pointer within the item. It is either 2 (over the icon), 4 (over the label), 8 (over the line), or 16 (over the symbol).

6.331 ItemDown Event 340

Applies To: [Section 6.348](#), [Section 6.629](#)

Description

If enabled, this event is reported when the user depresses a mouse button when the mouse pointer is over an item in a [Section 6.348](#) or [Section 6.629](#) object. This event is reported for information only and may not be controlled in any way using a callback function. Generating the event with `QDQ`, or calling it as a method, has no effect.

The event message reported as the result of `QDQ`, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ItemDown' or 340
[3]	Item number	Integer. The index of the item.
[4]	Mouse button	Integer.
[5]	Shift state	Integer. Sum of 1=shift key, 2=Ctrl key, 4=Alt key
[6]	Position	Integer. Indicates the position of the mouse-pointer within the item. It is either 2 (over the icon), 4 (over the label), 8 (over the line), 16 (over the symbol) or 32 (to the right of the label).

6.332 ItemGroupMetrics Property

Applies To: [Section 6.348](#)

Description

This property is used to specify colours and spacing elements for a [Section 6.348](#) that is displaying its Items in groupings (see [Section 6.333](#)).

****Note that this feature only applies if Native Look and Feel**

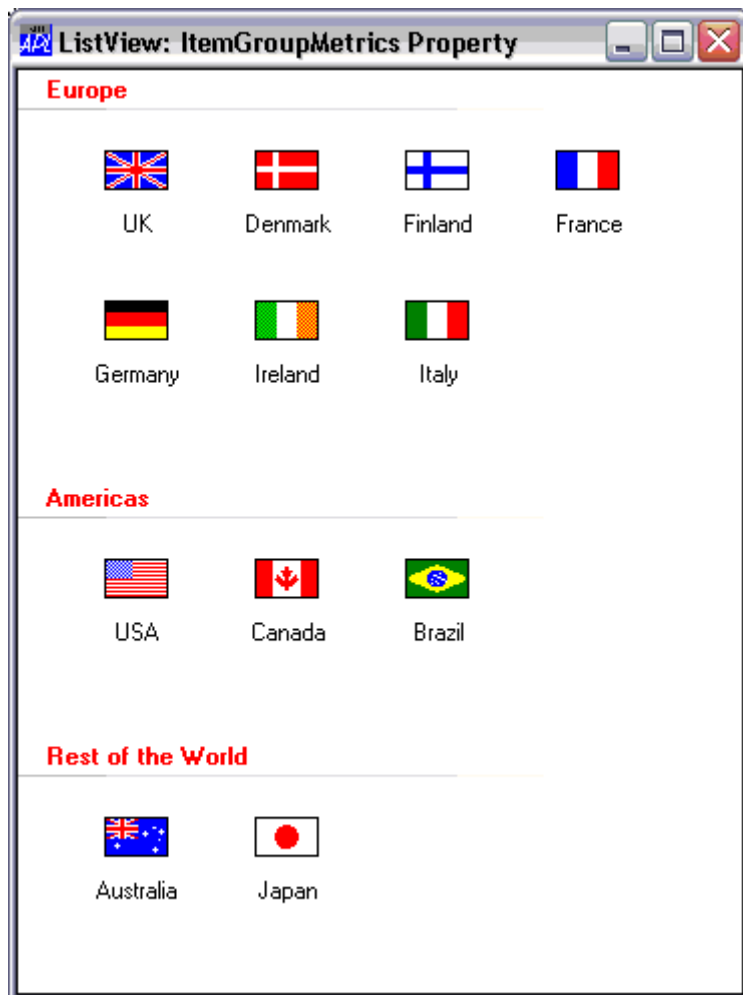
is enabled.**

ItemGroupMetrics is a 3-item nested vector as follows:

[1]	Text Colours	2-element vector of 3 element RGB values that specifies the colour of the group caption and group footer respectively.
[2]	Spacing	4-element integer vector that specifies the top, left, bottom and right spacing around each grouping in pixels
[3]	Border	Colours4-element vector of 3 element RGB values that specifies the colours for the top, left, bottom and right borders (not yet implemented).

The following expression, coupled with the code shown in the SetGroups example, causes the items to be displayed as shown below.

```
F.L.ItemGroupMetrics[1 2]←(2p<255 0 0)(10 100 0 10)
```



6.333 ItemGroups Property

Applies To: [Section 6.348](#)

Description

This property specifies item groupings for a [Section 6.348](#) object.

Note

This feature only applies if [Section 8.6](#) is enabled.

ItemGroups is a nested scalar or nested vector each of whose elements specifies a grouping. Each grouping is a 5-element vector as follows:

[1]	Group caption	character vector
[2]	Item index	Vector of indices to the Items property that specifies which Items are in this grouping.
[3]	Caption alignment	an integer: 1 = left aligned caption (the default) 2 = centre aligned caption 4 = right-aligned caption
[4]	State	Integer (not yet implemented)
[5]	Footer text	character vector (not yet implemented)

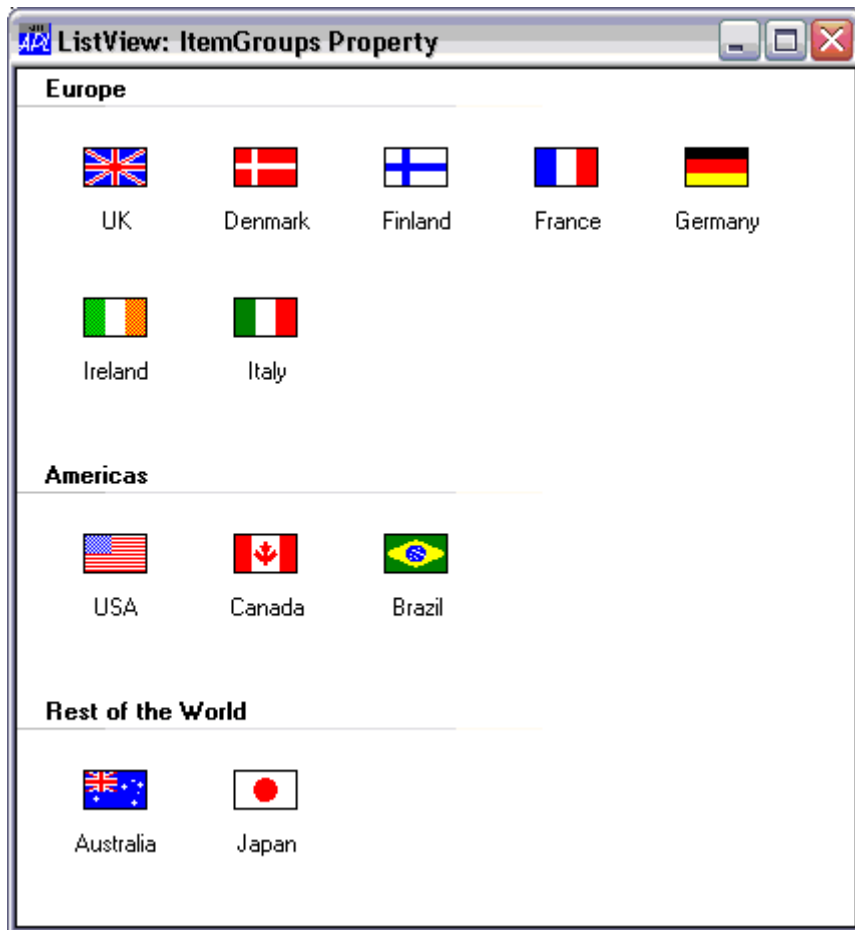
Note that State and Footer text are not yet implemented by Windows.

For example, the following expressions executed in the WTUTOR95 workspace will result in the display shown below.

```

'F'WC'Form' 'ListView Object'
II←ipCOUNTRIES
'F.L'WC'ListView'COUNTRIES(0 0)(100 100)
      ('ImageList' 'F.I1')
      ('ImageIndex' (ipCOUNTRIES))
'F.I1'WC'ImageList'('Size' 32 32)
(<'F.I1.')WC'(<'Icon' ''),,,↓↑FLAGBITS FLAGCMAP FLAGMASK
GROUPS←<'Europe'(11 4 5 6 7 8 9)
GROUPS,←<'Americas'(12 3 2)
GROUPS,←<'Rest of the World'(1 10)
F.L.ItemGroups←GROUPS

```



You can control the appearance of the groupings using the [Section 6.332](#) property.

6.334 Items Property

Applies To: [Section 6.121](#), [Section 6.122](#), [Section 6.346](#), [Section 6.348](#), [Section 6.563](#), [Section 6.629](#)

Description

This property specifies the list of items from which the user may choose.

The value of Items is a text array. It is normally specified as a vector of character vectors each of which represents an item. For a [Section 6.121](#), [Section 6.122](#), or [Section 6.346](#) or [Section 6.563](#), Items may also be a matrix whose rows specify items. If a character scalar or simple vector is specified, it is treated as a single item.

An empty character vector is treated the same as a vector of blanks, and represents one item.

A zero-length vector of vectors or an empty matrix represents 0 items. The default value for Items is an empty matrix.

`⎕WG 'Items'` returns an array of the same structure as was assigned by *Dyalog APL Language*: *Wc* or *Dyalog APL Language*: *Ws*.

6.335 ItemUp Event 341

Applies To: [Section 6.348](#), [Section 6.629](#)

Description

If enabled, this event is reported when the user releases a mouse button when the mouse pointer is over an item in a [Section 6.348](#) or [Section 6.629](#) object. This event is reported for information only and may not be controlled in any way using a callback function. Generating the event with `⎕NQ`, or calling it as a method, has no effect.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ItemUp' or 341
[3]	Item number	Integer. The index of the item.
[4]	Mouse button	Integer.
[5]	Shift state	Integer. Sum of 1=shift key, 2=Ctrl key, 4=Alt key
[6]	Position	Integer. Indicates the position of the mouse-pointer within the item. It is either 2 (over the icon), 4 (over the label), 8 (over the line), or 16 (over the symbol).

6.336 Justify Property

Applies To: [Section 6.55](#), [Section 6.56](#), [Section 6.185](#), [Section 6.341](#), [Section 6.563](#), [Section 6.585](#)

Description

This property determines the manner in which text is justified within the object. It is a character vector that may take the value 'Left' (the default), 'Centre' or 'Right'. The keyword 'Centre' may also be spelled 'Center'.

When applied to an [Section 6.185](#) object with [Section 6.577](#) 'Multi', a value of 'Centre' or 'Right' forces word-wrapping and disables horizontal scrolling. Note that Justify only applies to a multi-line edit field. If you specify a value for Justify in a 1-line edit field ([Section 6.577](#) 'Single'), it will be ignored.

For a [Section 6.585](#), Justify may be 'Right' (which is the default) or 'None' or empty.

If Justify is 'Right', the [Section 6.585](#) increases the width of each tab, if necessary, so that each row of tabs fills the entire width of the tab control. Otherwise, if Justify is empty or 'None', the rows are ragged.

With the exception of [Section 6.341](#) and [Section 6.585](#) objects, Justify may only be specified when the object is created using `WC`.

6.337 KeepBits Property

Applies To: [Section 6.47](#), [Section 6.136](#), [Section 6.309](#)

Description

This property is be used to control the way that [Section 6.47](#), [Section 6.136](#) and [Section 6.309](#) objects are stored in the workspace.

When you create a [Section 6.47](#), [Section 6.309](#) or [Section 6.136](#) using `□WC`, APL asks Windows to allocate a corresponding bitmap, icon or cursor *resource*. This resource is allocated in Windows memory. If APL were to hold the values of the *image properties* ([Section 6.68](#), [Section 6.48](#) and [Section 6.104](#) for a [Section 6.47](#); [Section 6.48](#), [Section 6.104](#) and [Section 6.366](#) for [Section 6.136](#) and [Section 6.309](#) objects) internally in the workspace, this data would be duplicated. For large bitmaps this would have a serious impact on memory utilisation and may affect performance. The `KeepBits` property is provided to allow you to control whether or not APL retains the values of the *image properties* in the workspace, so that you can choose a strategy to suit your configuration and requirements. `KeepBits` may take the value 0 or 1.

If `KeepBits` is 0 the values of the *image properties* are **not** stored internally in your workspace. If you save a workspace containing a [Section 6.47](#), [Section 6.136](#) or [Section 6.309](#) object, the corresponding Windows resource is automatically re-allocated when the workspace is loaded by referring to the associated file. This is the file whose full pathname is defined by the value of the object's [Section 6.207](#) property. It follows that if you adopt this strategy, you must ensure that the [Section 6.207](#) property is set correctly. If APL cannot find the file when the workspace is `)LOADED`, it cannot re-create the object, and you will get a `VALUE ERROR` when you subsequently refer to it. A further consideration is the effect on `□WG`. If `KeepBits` is 0, and you execute `□WG 'CBits'` or `'Bits'` or `'CMap'` or `'Mask'`, APL obtains these values by requesting the data from Windows.

If `KeepBits` is set to 1, the contents of the *image properties* are stored in the workspace, thus duplicating the information which is held by Windows itself. If you save a workspace containing a [Section 6.47](#), [Section 6.136](#) or [Section 6.309](#) the corresponding Windows resource is automatically re-allocated from the *image properties* when the workspace is loaded. The value of the [Section 6.207](#) property is ignored. When you execute `□WG 'CBits'` or `'Bits'` or `'CMap'` or `'Mask'`, APL generates the result directly from the stored values held (internally) in the workspace.

6.338 KeepOnClose Property

Applies To

Description

This property is either 0 or 1 and determines how the object is treated when its parent Form (or, in the case of a Form, the Form itself) is closed by the user, receives a Close event from `⌈NQ`, or when Close is called as a method.

If `KeepOnClose` is 1 (for the object itself **and** for all its parents) when its parent Form is closed, the object changes from being a GUI object to a pure namespace. For example, the Type of a Button will change from 'Button' to 'Namespace'. Effectively, the GUI component of the object is discarded but its Namespace component (and any variables, functions, operators and other namespaces that it contains) remains intact. Monadic `⌈WC` may subsequently be used to re-attach the GUI component to the object. All child GUI objects are treated in the same way.

Note that the default value of `KeepOnClose` depends upon the way in which a GUI object was created with `⌈WC`. If a GUI object is created by dyadic `⌈WC`, `KeepOnClose` defaults to 0. If a GUI object is attached by monadic `⌈WC`, its `KeepOnClose` property defaults to 1.

Note

The use of `KeepOnClose` is deprecated; Dyalog may at some point remove this Property or disable its functionality.

6.339 KeyError Event 23

Applies To: [Section 6.56](#), [Section 6.185](#), [Section 6.563](#)

Description

If enabled, this event is generated when the user presses and releases a key on the keyboard that is invalid for the FieldType of the object and has been ignored. This event is reported for information only and you may not disable it or modify it in any way.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'KeyError' or 23
[3]	Character	character scalar
[4]	Character code	integer scalar
[5]	Key Number	integer scalar
[6]	Shift state	integer scalar

In the Classic Edition, the resolution of the keystroke to a character (in □AV) is performed using the Input Translate Table. In the Unicode Edition, the resolution is performed by the Operating System.

In the Unicode Edition, the Character Code is the Unicode code point of the character that the user entered. In the Classic Edition, it is a number in the range 0-255 which specifies the ASCII character that would normally be generated by the keystroke, and is independent of the Input Translate Table. If there is no corresponding ASCII character, the ASCII code reported is 0.

The key number is the physical key number reported by Windows when the key is pressed.

The Shift State indicates which (if any) of the Shift, Ctrl and Alt keys are down at the same time as the key is pressed. It is the sum of the following numbers :

Thus a Shift State of 3 indicates that the user has pressed the key in conjunction with both the Shift and Ctrl keys. A Shift State of 0 indicates that the user pressed the key on its own.

6.340 KeyPress Event 22

Applies To

Description

If enabled, this event is generated when the user presses and releases a key on the keyboard. It is reported for whichever object has the keyboard focus at the time.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'KeyPress' or 22
[3]	Input Code	character scalar or vector
[4]	Character code	integer scalar
[5]	Key Number	integer scalar
[6]	Shift state	integer scalar

If the keystroke resolves to a character, the Input Code is a character scalar.

If the keystroke resolves to a command recognised by Dyalog APL, such as UC (Up Cursor) or ER (Enter) the Input Code contains the corresponding 2-element character vector.

In the Classic Edition, the resolution of the keystroke to a character (in □AV) or to a command, is performed using the Input Translate Table.

In the Unicode Edition, the resolution is performed by the Operating System. However, if the keystroke resolves to a navigation or control key (such as Cursor Up or Enter), the same 2-character "command" is reported. Note however that commands that are purely internal to Dyalog APL (such as Trace, commonly Ctrl+Enter) are not reported as such and the Input Code will be empty.

In the Unicode Edition, the Character Code is the Unicode code point of the character that the user entered. In the Classic Edition, it is a number in the range 0-255 which specifies the ASCII character that would normally be generated by the keystroke, and is independent of the Input Translate Table. If there is no corresponding ASCII character, the ASCII code reported is 0.

The key number is the physical key number reported by Windows when the key is pressed.

The Shift State indicates which (if any) of the Shift, Ctrl and Alt keys are down at the same time as the key is pressed. It is the sum of the following numbers :

1	Shift key down
2	Control key down
4	Alt key down

Thus a Shift State of 3 indicates that the user has pressed the key in conjunction with both the Shift and Ctrl keys. A Shift State of 0 indicates that the user pressed the key on its own.

EXAMPLE

```

▼ Key;Form1
[1] 'Form1'WC'Form'('Event' 'KeyPress' 'Keycb')
[2] DQ'Form1'
▼

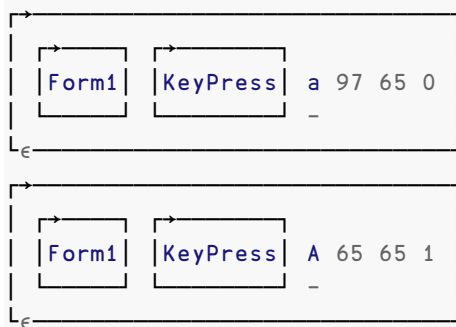
▼ Keycb msg
[1] DISPLAY msg
▼

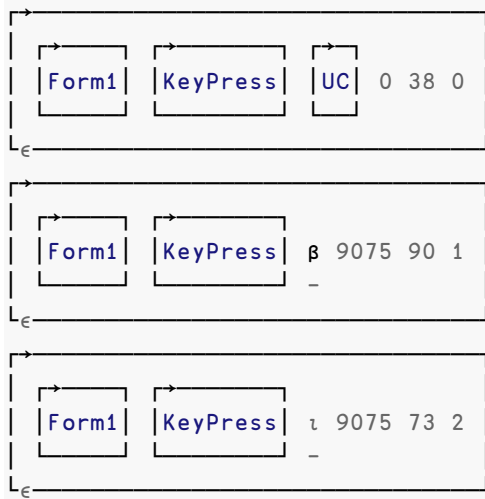
```

On running function Key, the following output will be displayed as a result of the user pressing the following 5 keys in succession:

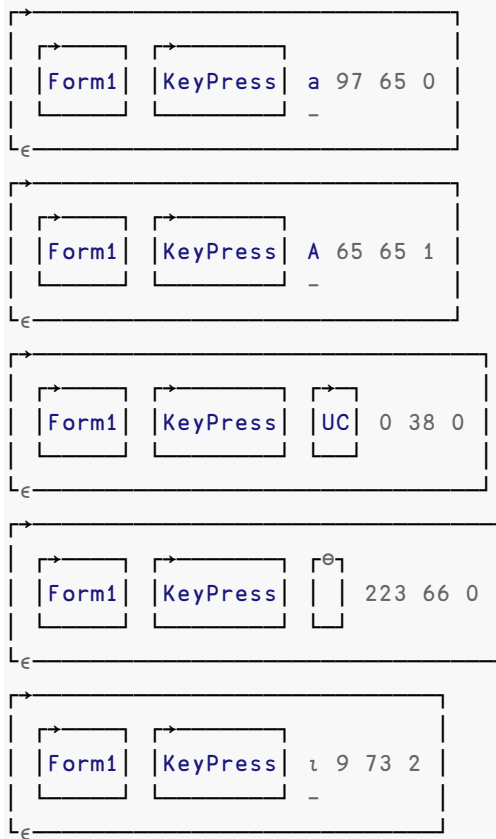
1. "a"
2. Shift+"a"
3. Cursor Up
4. β ("b" using a Greek keyboard)
5. ι (Ctrl+"i" using a UK APL keyboard)

Unicode Edition





Classic Edition



6.341 Label Object

Purpose: Displays static text.

Parents ActiveXControl CoolBand Form Grid Group PropertyPage SubForm ToolBar ToolControl

Children Circle Cursor Ellipse Font Marker Poly Rect Text Timer

Properties Type Caption Posn Size Coord Border Justify Active Visible Event Sizeable Draggable FontObj FCol BCol CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj FieldType Decimals FormatString Value Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ChooseFont

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel Select

Description

This object displays a text label, a number, a date or a time value.

If [Section 6.206](#) is empty, the Label displays the text defined by its [Section 6.66](#) property. If the [Section 6.66](#) property contains one or more linefeed characters (␣UCS 10) the Label becomes a multi-line Label which is top-left justified and automatically wraps on white-space characters (such as space and tab) to fit in the width provided.

If [Section 6.206](#) is 'Numeric', 'LongNumeric', 'Currency', 'Date', 'LongDate', or 'Time' the Label converts and formats the number defined by its [Section 6.641](#) property and displays this instead. See [Section 6.206](#) property for details.

The [Section 6.49](#) property determines whether or not the label has a border. A value of 0 means no border (the default). A value of 1 means that a 1-pixel border is drawn around the label.

By default, the value of the [Section 6.184](#) property for a Label is 'None' and the value of [Section 6.45](#) is 0 which is normally white, or grey if the parent object has a 3-dimensional appearance. You can change its appearance by setting [Section 6.184](#) and/or [Section 6.45](#) to different values.

6.342 LastError Property

Applies To: [Section 6.7](#), [Section 6.414](#), [Section 6.420](#), [Section 6.489](#)

Description

The LastError property provides information about the most recent error reported by OLE. You may use this property to report an error from an [Section 6.420](#) or [Section 6.7](#) to a host application.

6.343 LateBind Property

Applies To: [Section 6.414](#), [Section 6.420](#)

Description

LateBind is a Boolean property that determines whether or not the Type Information is read when an [Section 6.414](#) object is instantiated, or when an [Section 6.420](#) references a COM object. It must be set when the [Section 6.414](#) or [Section 6.420](#) object is created and cannot subsequently be changed.

If LateBind is 0 (the default) the Type information is obtained, in its entirety, when the [Section 6.414](#) object is instantiated or when an [Section 6.420](#) processes a reference to an external COM object.

If LateBind is 1, APL postpones the calls to obtain Type Information until a particular property, method or event of the COM object is referenced; and then only for that particular member.

6.344 LicenseKey Property

Applies To: [Section 6.412](#)

Description

The LicenseKey property is a character string that contains the license key for an ActiveX control.

If an ActiveX control requires a license key, it must be specified by an application when it creates an instance of the control. Typically, the license key is *required* only by the run-time version of an ActiveX control, and is *made available* to an application by the development version of the control.

6.345 Limits Property

Applies To: [Section 6.459](#), [Section 6.508](#), [Section 6.563](#), [Section 6.625](#), [Section 6.637](#)

Description

This property is a 2-element vector that specifies the minimum and maximum values of an object. The values must be in the range (-2×31) to $(-1 + 2 \times 31)$ inclusive.

6.346 List Object

Purpose: Allows the user to select one or more items from a list.

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm ToolBar ToolControl

Children Circle Cursor Ellipse Font Marker Poly Rect Text Timer

Properties Type Items Posn Size Style Coord Border Active Visible Event VScroll SellItems Sizeable Dragable FontObj FCol BCol CursorObj AutoConf Index Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles MultiColumn ColumnWidth KeepOnClose Redraw SortItems TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ChooseFont

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap Select

Description

The [Section 6.334](#) property is either a vector of character vectors or a character matrix, and determines the items in the List.

The size and position of the area used to display the list is defined by [Section 6.556](#) and [Section 6.452](#). If [Section 6.556](#) is not chosen to represent an exact number of lines of text, the bottom line of text may be clipped.

The [Section 6.319](#) property specifies or reports the position of [Section 6.334](#) in the list box as a positive integer value. If [Section 6.319](#) has the value "n", it means that the "nth" item in [Section 6.334](#) is displayed on the top line in the list box. However, it is

ignored if all the [Section 6.334](#) fit within the List object. Note that [Section 6.319](#) can only be set using *Dyalog APL Language: Ws* and not by *Dyalog APL Language: Wc*. The default value for [Section 6.319](#) is `⍺IO`.

The [Section 6.577](#) property may be 'Single' (the default) or 'Multi'. 'Single' allows only a single item to be selected. 'Multi' allows several items to be chosen. In either case, if the [Section 6.513](#) event is enabled, it is generated whenever the selection changes. If [Section 6.577](#) is 'Multi' the List will generate a [Section 6.513](#) event every time an item is added to the selected list.

Under Windows, you may select or de-select multiple items in a List object by pressing the Ctrl key at the same time as pressing the left mouse button.

The [Section 6.519](#) property is a Boolean vector with one element per element or row in [Section 6.334](#) and indicates which (if any) of the items is currently selected (and highlighted).

The [Section 6.646](#) property determines whether or not the list has a scrollbar. Its possible values are :

-2	scrollbar if required
-1	scrollbar if required
0	no scrollbar

Note that data in a List is always scrollable if there are more items than will fit in the box. [Section 6.646](#) determines ONLY whether or not a scrollbar is provided.

The [Section 6.403](#) property is a Boolean value that specifies whether or not the List object displays its items in columns. The default is 0 which produces a single-column display. If [Section 6.403](#) is 1, the List object displays its items in columns whose width is defined by the [Section 6.120](#) property.

6.347 ListTypeLibs Method 520

Applies To: [Section 6.489](#)

Description

The ListTypeLibs method reports the names and CLSIDs of all the loaded Type Libraries.

The ListTypeLibs method is niladic.

The result is a nested vector with one element per loaded Type Library.

Each element is a vector of 2-element character vectors. The first is the name of the Type Library; the second is its class identifier or CLSID.

EXAMPLE

```
'EX'WC'OLEClient' 'Excel.Application'
ρListTypeLibs
3
↑>ListTypeLibs
Microsoft Excel 9.0 Object Library
{00020813-0000-0000-C000-000000000046}

↑>"ListTypeLibs
Microsoft Excel 9.0 Object Library
Microsoft Visual Basic for Applications Extensibility 5.3
Microsoft Office 9.0 Object Library
```

6.348 ListView Object

Purpose: The ListView object displays a collection of items.

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm ToolBar
ToolControl

Children Bitmap Circle Cursor Ellipse Font Icon ImageList Marker Poly Rect Text Timer

Properties Type Items Posn Size Style Coord Align Border Active Visible Event
DragItems View AutoArrange Header Wrap EditLabels ImageListObj ReportInfo ColTitles
ImageIndex ReportImageIndex ReportBCol HeaderImageList HeaderImageIndex VScroll
HScroll SellItems Sizeable Draggable FontObj FCol BCol CursorObj AutoConf Data Attach
EdgeStyle Handle Hint HintObj Tip TipObj ColTitleAlign ColTitle3D Translate Accelerator
AcceptFiles KeepOnClose CheckBoxes FullRowSelect GridLines Redraw TabIndex
AlwaysShowSelection ItemGroups ItemGroupMetrics MethodList ChildList EventList
PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj
GetItemState SetItemState GetItemPosition

Events Close Create FontOK FontCancel BeginEditLabel EndEditLabel DragDrop
Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus
LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter
MouseLeave MouseWheel GestureZoom GesturePan GestureRotate

GestureTwoFingerTap GesturePressAndTap ColumnClick SetItemPosition ItemDown
ItemUp ItemDbClick GetTipText SetColSize Select

Description

The ListView object is a window that displays a collection of items, each item consisting of an icon and a label. The ListView provides several ways of arranging items and displaying individual items. For example, additional information about each item can be displayed in columns to the right of the icon and label. An example of the use of a ListView object is the "My Computer" Windows utility.

The Items property is a vector of character vectors that specifies the labels for the items displayed by the ListView. The [Section 6.317](#) property specifies the names of two [Section 6.316](#) objects that define two sets of icons; a large icon (32x32 pixel) set and a small icon (16x16 pixel) set. Alternatively, [Section 6.317](#) may be empty (no icons displayed) or contain just the name of a single large icon [Section 6.316](#).

The [Section 6.644](#) property contains a character vector that determines how the items are displayed. It may have one of the following values; 'Icon' (the default), 'SmallIcon', 'List' or 'Report'. When View is 'Icon' or 'SmallIcon', the items are arranged *row-wise* with large or small icons as appropriate. When View is set to 'List', the items are arranged *column-wise* using small icons. Examples of 'Icon' and 'List' views are illustrated below.

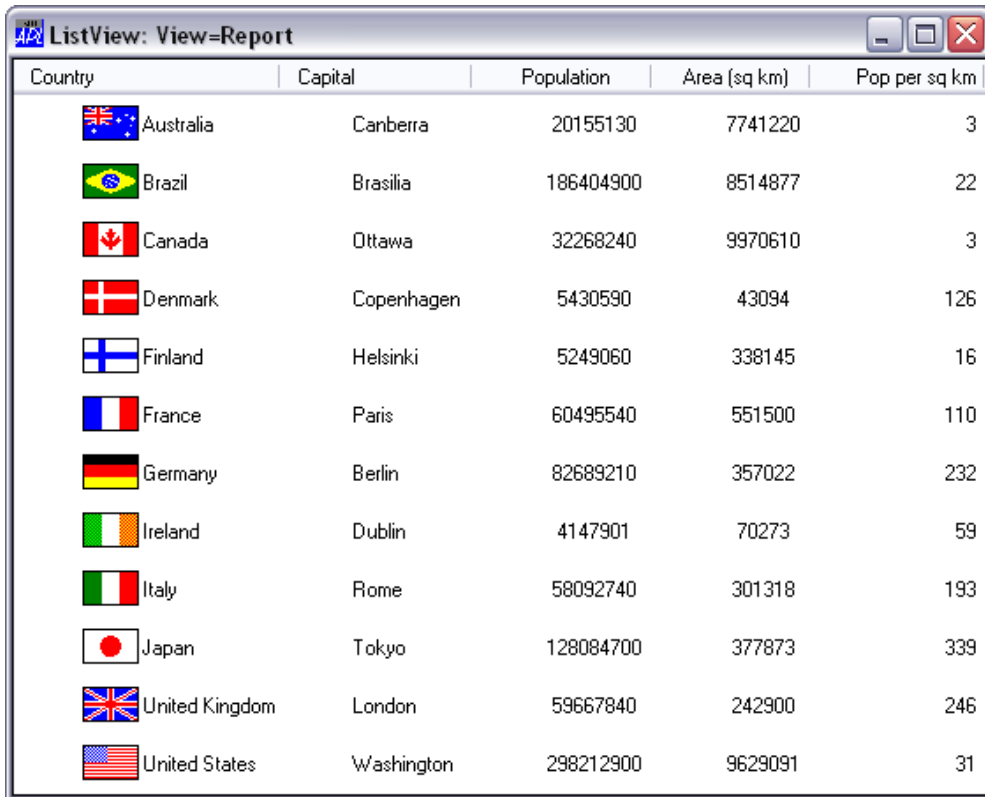




When View is set to 'Report', the items are displayed in a single column using small icons but with the matrix specified by [Section 6.480](#) displayed alongside. In this format, the Boolean [Section 6.291](#) property determines whether or not the object also provides column headings. Its default value is 1. The column headings themselves are specified by the ColTitles property. Their alignment (and the alignment of the data in the columns beneath them) is defined by the [Section 6.114](#) property.

The appearance of the column titles is further controlled by the [Section 6.113](#) property. This is a Boolean value (default 1) which specifies whether or not the column titles have a 3-dimensional (plinth) appearance. [Section 6.291](#) and [Section 6.113](#) may only be set when the object is created using `WC` and may not subsequently be changed by `WS`.

In 'Report' View, columns may be resized by the user dragging the bars between the titles, or under program control using the SetColSize event. A 'Report' view example is illustrated below.



Country	Capital	Population	Area (sq km)	Pop per sq km
 Australia	Canberra	20155130	7741220	3
 Brazil	Brasilia	186404900	8514877	22
 Canada	Ottawa	32268240	9970610	3
 Denmark	Copenhagen	5430590	43094	126
 Finland	Helsinki	5249060	338145	16
 France	Paris	60495540	551500	110
 Germany	Berlin	82689210	357022	232
 Ireland	Dublin	4147901	70273	59
 Italy	Rome	58092740	301318	193
 Japan	Tokyo	128084700	377873	339
 United Kingdom	London	59667840	242900	246
 United States	Washington	298212900	9629091	31

The [Section 6.175](#) property is Boolean and specifies whether or not the user may drag an item from one position to another. Its default value is 1 (dragging is enabled).

The [Section 6.33](#) property is Boolean and specifies whether or not the items are automatically re-arranged whenever an item is repositioned by the user or moved under program control. Its default value is 0.

The [Section 6.188](#) is a Boolean property (default 0) that determines whether or not the user may edit the labels which are specified by the Items property.

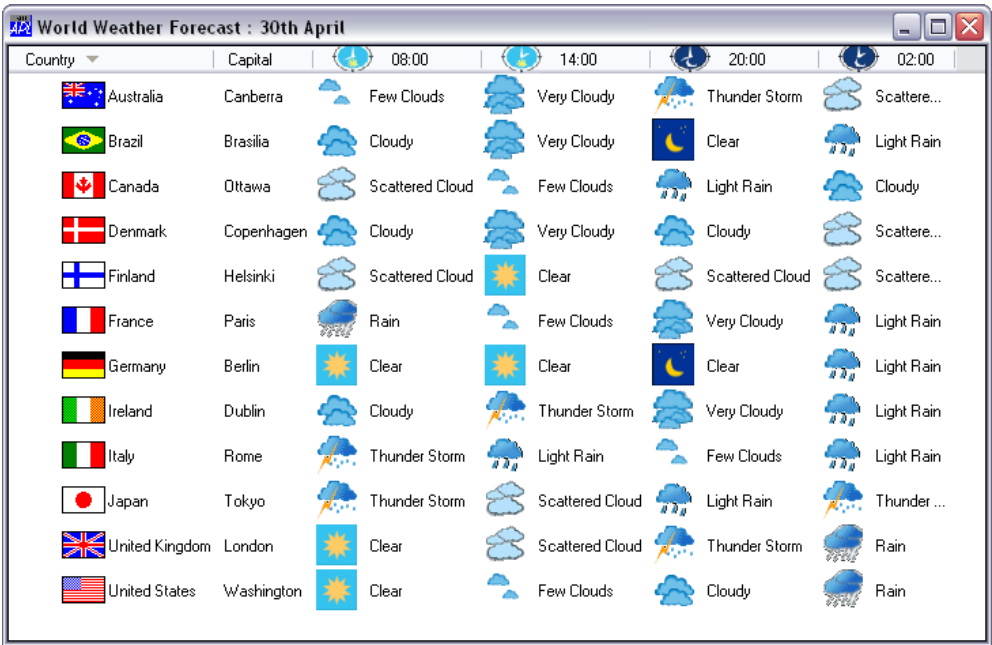
The [Section 6.577](#) property may be 'Single', which specifies that only one item may be selected at a time, or 'Multi' which permits multiple selections to be made. The default is 'Multi'.

The [Section 6.89](#) property is Boolean and specifies whether or not check boxes are drawn to the left of items. Its default value is 0.

The [Section 6.273](#) property is Boolean and specifies whether or not grid lines are drawn between items. This applies only when View is 'Report'. Its default value is 0.

In 'Report' View, the [Section 6.479](#) property specify images to be displayed alongside each item, and overrides the images specified by [Section 6.317](#). The [Section 6.293](#) and [Section 6.292](#) properties specify images to be displayed alongside each column title.

The following example illustrates the use of these properties.



The screenshot shows a window titled "World Weather Forecast : 30th April". It contains a table with columns for Country, Capital, and four time slots: 08:00, 14:00, 20:00, and 02:00. Each cell in the table displays a weather icon and a text description. The rows represent different countries, including Australia, Brazil, Canada, Denmark, Finland, France, Germany, Ireland, Italy, Japan, United Kingdom, and United States.

Country	Capital	08:00	14:00	20:00	02:00
Australia	Canberra	Few Clouds	Very Cloudy	Thunder Storm	Scattere...
Brazil	Brasilia	Cloudy	Very Cloudy	Clear	Light Rain
Canada	Ottawa	Scattered Cloud	Few Clouds	Light Rain	Cloudy
Denmark	Copenhagen	Cloudy	Very Cloudy	Cloudy	Scattere...
Finland	Helsinki	Scattered Cloud	Clear	Scattered Cloud	Scattere...
France	Paris	Rain	Few Clouds	Very Cloudy	Light Rain
Germany	Berlin	Clear	Clear	Clear	Light Rain
Ireland	Dublin	Cloudy	Thunder Storm	Very Cloudy	Light Rain
Italy	Rome	Thunder Storm	Light Rain	Few Clouds	Light Rain
Japan	Tokyo	Thunder Storm	Scattered Cloud	Light Rain	Thunder ...
United Kingdom	London	Clear	Scattered Cloud	Thunder Storm	Rain
United States	Washington	Clear	Few Clouds	Cloudy	Rain

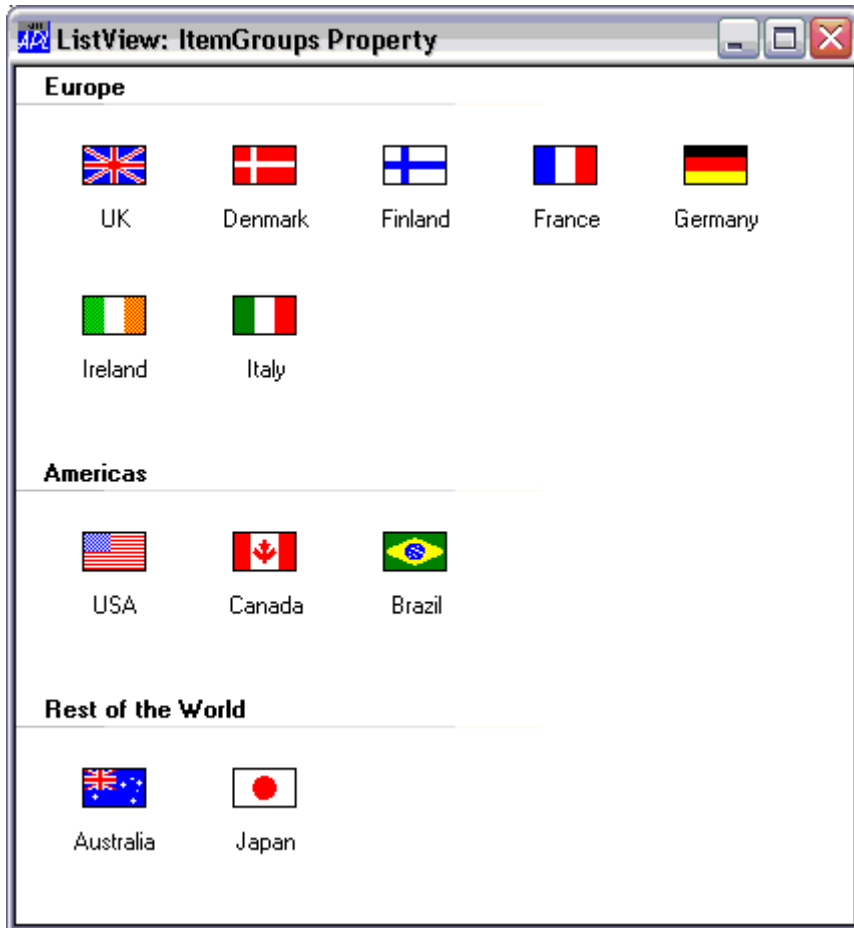
The [Section 6.478](#) property specifies each item's background colour.

The [Section 6.232](#) property is Boolean and specifies whether or not the entire row is highlighted to indicate selected items. This applies only when View is 'Report'. Its default value is 0.

The [Section 6.333](#) and [Section 6.332](#) properties allow you to display items in groups as illustrated below.

Note

This feature only applies if [Section 8.6](#) is enabled.



6.349 LoadEnd Event 836

Applies To: [Section 6.307](#)

Description

A LoadEnd event is raised when a particular frame has finished loading. Multiple frames may be loading at the same time. Sub-frames may start or continue to load even after the main frame has finished loading.

A common technique is to wait for the main frame to finish loading before further interaction with the HTMLRenderer instance. In this case, you should set up an event handler on the LoadEnd event and check the 4th element which indicates if the loaded frame is the main frame.

You may use the [Section 6.328](#) property to check if the HTMLRenderer is still loading.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'LoadEnd' or 836
[3]	url	The URL of the loaded frame
[4]	Flag	1 if the loaded frame is the "main" frame, 0 otherwise
[5]	Code	The HTTP status code as a result of loading the frame

6.350 LocalAddr Property

Applies To: [Section 6.604](#)

Description

The LocalAddr property is a character vector that specifies the IP address of your computer. Its default value is '0.0.0.0'.

Unless your computer has more than one network adapter each identified by a different IP address, you do not need to specify LocalAddr. However, in this case you may use *either* LocalAddr *or* [Section 6.351](#) to identify the adapter. If you specify both properties, the value of [Section 6.351](#) will be ignored.

Note that you may also set the value of LocalAddr to an empty character vector. In this case, the value returned by `⌈WG` will be '0.0.0.0'.

LocalAddr may only be specified in the `⌈WC` statement that creates the [Section 6.604](#) and may not subsequently be changed using `⌈WS`.

6.351 LocalAddrName Property

Applies To: [Section 6.604](#)

Description

The LocalAddrName property is a character vector that specifies the host name of your computer. It may be useful when you have more than one network adapter (perhaps an

Ethernet adapter and a token ring adapter) and you wish to avoid hard-coding the IP address.

Note that you may use *either* [Section 6.350](#) or LocalAddrName to identify the local computer. If you specify both properties, the value of LocalAddrName will be ignored.

LocalAddrName may only be specified by a server [Section 6.604](#). Furthermore, it must be specified in the `□WC` statement that creates the [Section 6.604](#) object and it may not subsequently be changed using `□WS`.

When the specified host name has been resolved to an IP address, the [Section 6.604](#) will generate a [Section 6.598](#) event and update the value of [Section 6.350](#) accordingly.

For a client [Section 6.604](#), you may not specify LocalAddrName and `□WC` returns an empty character vector.

6.352 Locale Property

Applies To: [Section 6.414](#)

Description

The Locale property specifies the language in which the OLE server, attached to an OLEClient, exposes its methods (functions) and properties (variables).

When you create an OLEClient object, Dyalog APL/W requests the default Type Library associated with the OLE server that you specify. Many OLE servers, such as Excel.Application, provide different names for the methods and properties they expose for different languages. Without Locale, it would be difficult to write an OLE client application that could run in different countries, as the names of the functions and variables may be unpredictable.

Locale is an integer; for example, the value 9 specifies English and the value 12 specifies French.

Locale may **only** be specified by the `□WC` statement that is used to create the OLEClient; it may not subsequently be changed using `□WS`. A table of commonly used Locale values is given below.

Note that Dyalog cannot guarantee that you will actually be given the Locale you specify. This is a function of your specific installation and the OLE server in question. However, Dyalog believes that for Microsoft products, it is a fairly safe bet that the US/English interface will be available in most countries.

Language	Locale
Neutral	0
Danish	6
Dutch	19
English	9
Finnish	11
French	12
German	7
Italian	16
Norwegian	20
Portuguese	22
Russian	25
Spanish	10
Swedish	29

6.353 LocalPort Property

Applies To: [Section 6.604](#)

Description

The LocalPort property is a scalar integer in the range 1-65536 that identifies the [Section 8.7](#) associated with a [Section 6.604](#) object.

Note that you may use *either* LocalPort or [Section 6.354](#) to identify the service. The use of [Section 6.354](#) is slightly slower but it avoids hard-coding the port number in your program and is generally more flexible. If you specify both properties, the value of [Section 6.354](#) will be ignored.

LocalPort may be specified only by the process that is initiating the connection (the server) and must be set by the `□WC` statement that creates the [Section 6.604](#). LocalPort may not subsequently be changed using `□WS`

If you specify a value of 0, the system will assign an available port number. For example:

```
'S1' □WC'TCPSocket' ( 'LocalPort' 0)
S1.LocalPort
4047
```

For a process that is completing a connection, LocalPort is allocated by the system and is effectively read-only.

6.354 LocalPortName Property

Applies To: [Section 6.604](#)

Description

The LocalPortName property is a character vector that specifies the port name of the local service that you wish to offer as a server.

Note that you may use *either* [Section 6.353](#) or LocalPortName to identify the service. The use of LocalPortName is slightly slower but it avoids hard-coding the port number in your program and is generally more flexible. If you specify both properties, the value of LocalPortName will be ignored.

LocalPortName may be specified only by the process that is initiating the connection (the server) and must be set by the □WC statement that creates the [Section 6.604](#). LocalPortName may not subsequently be changed using □WS.

When the specified port name has been resolved to a port number, the [Section 6.604](#) will generate a [Section 6.599](#) event and update the value of [Section 6.353](#) accordingly.

For a client [Section 6.604](#), you may not specify LocalPortName and □WG returns an empty character vector.

6.355 Locator Object

Purpose: Allows the user to input a point, ellipse, line or rectangle.

Parents ActiveXControl Form Group PropertyPage PropertySheet Root Static SubForm
TCPSocket ToolBar ToolControl

Children Timer

Properties Type Posn Size LStyle Style Coord Event Step Sizeable CursorObj Data
Accelerator KeepOnClose MethodList ChildList EventList PropList

Methods Detach Wait**Events** Close Create Locator Select**Description**

This object is used to obtain graphical input from the user. Like a pop-up menu or a [Section 6.399](#), the Locator is a *modal* object whose interaction with the user is initiated by a "local" *Dyalog APL Language: Dq*.

This is terminated when the user releases a mouse button or presses any key other than a cursor movement key, Shift, Ctrl or Alt. It is usual to initiate the *Dyalog APL Language: Dq* for the Locator from within a callback function attached to a [Section 6.392](#) (1) [Section 6.196](#).

When the "local" *Dyalog APL Language: Dq* is terminated, a [Section 6.356](#) (80) [Section 6.196](#) is generated. The associated event message contains the new position and size of the Locator, together with how the event was generated (keystroke or mouse button). To obtain the Locator's new position or size, you **must** enable the event by setting its "action" code to 1, or to the name of a suitable callback function.

The value of the [Section 6.577](#) property determines the type of locator displayed. It may be 'Point', 'Line', 'Rect', or 'Ellipse'. The default value is 'Rect'. The value of the [Section 6.557](#) property is 0 or 1 and determines whether or not "rubberbanding" is enabled. Its default value is 1 which turns "rubberbanding" on. The [Section 6.556](#) property determines the initial size of the Locator when displayed by *Dyalog APL Language: Dq*. Its default value is (0,0).

If [Section 6.577](#) is 'Rect' the Locator displays a rectangle. One corner of the rectangle is positioned at [Section 6.452](#). The diagonally opposite corner is positioned at ([Section 6.452](#)+[Section 6.556](#)). If [Section 6.557](#) is 0, the entire rectangle is dragged as the mouse is moved. If [Section 6.557](#) is 1, the

corner initially defined by ([Section 6.452](#)+[Section 6.556](#)) is dragged (rubberbanding the rectangle) as the mouse is moved. The rectangle disappears when the operation is terminated. The new position or size of the rectangle is reported in the [Section 6.356](#) event message.

If [Section 6.577](#) is 'Ellipse' the Locator displays an ellipse. One corner of the bounding rectangle of the ellipse is positioned at [Section 6.452](#). The diagonally opposite corner is positioned at ([Section 6.452](#)+[Section 6.556](#)). If [Section 6.557](#) is 0, the entire ellipse is dragged as the mouse is moved. If [Section 6.557](#) is 1, the corner of the bounding rectangle initially defined by ([Section 6.452](#)+[Section 6.556](#)) is dragged (rubberbanding the ellipse) as the mouse is moved. The ellipse disappears when the

operation is terminated. The new position or size of the bounding rectangle of the ellipse is reported in the [Section 6.356](#) event message.

If [Section 6.577](#) is 'Line' the Locator displays a line drawn between the points defined by [Section 6.452](#) and [Section 6.452+Section 6.556](#). If [Section 6.557](#) is 0, the line is dragged with the cursor as the mouse is moved. If [Section 6.557](#) is 1, the end of the line initially defined by [Section 6.452+Section 6.556](#) is dragged (rubberbanding the line) as the mouse is moved. The line disappears when the operation is terminated. The new position or size of the line is reported in the [Section 6.356](#) event message.

If 'Style' is 'Point', the values of [Section 6.557](#) and [Section 6.556](#) are ignored. During the *Dyalog APL Language: Dq* no visible feedback (other than the cursor) is provided as the user moves the mouse. When the *Dyalog APL Language: Dq* terminates, the new position of the Locator is reported in the [Section 6.356](#) event message.

The [Section 6.576](#) property is a 2-element integer vector (default value 1 1) that specifies the increments (in pixels) by which the size or position of the Locator changes in the Y and X directions respectively as the user moves the Locator.

The Locator is normally initiated from a [Section 6.392](#) (1) event, and it is natural to place it at the current cursor position. However, if you are using rubberbanding, you will normally want to have the cursor appear at the end or corner of the Locator that moves. If you start with a non-zero sized Locator, you must set [Section 6.452](#) (which defines the **fixed** end or corner) to the current cursor position minus [Section 6.556](#) to achieve this effect.

6.356 Locator Event 80

Applies To: [Section 6.355](#)

Description

If enabled, this event is generated when the user releases a mouse button, or presses any key (other than a cursor movement key) during a *Dyalog APL Language: Dq* on a [Section 6.355](#) object.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 9-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Locator' or 80
[3]	Y	y-position of <u>Locator</u> after <i>Dyalog APL Language: Dq</i>
[4]	X	x-position of <u>Locator</u> after <i>Dyalog APL Language: Dq</i>
[5]	H	height of <u>Locator</u> after <i>Dyalog APL Language: Dq</i>
[6]	W	width of <u>Locator</u> after <i>Dyalog APL Language: Dq</i>
[7]	Mouse Button	number of the button which was released (0 if keystroke)
[8]	Keystroke	character scalar or vector containing the "Input Code" for the key that terminated the operation
[9]	Shift state	integer scalar

6.357 LockColumns Method 227

Applies To: [Section 6.263](#)

Description

This method is used to lock one or more columns of a [Section 6.263](#) object. However, LockColumns is not supported in combination with hierarchical column titles as specified by the [Section 6.116](#) property.

The argument to LockColumns is a 1 or 2-element vector as follows.

[1]	Column(s)	integer scalar, vector or matrix
[2]	Lock flag	0 or 1

Column(s) may be a scalar or a vector specifying the column or columns to be locked or unlocked. Alternatively, it may be a matrix whose first row specifies the columns to be locked and whose second row specifies *where* they are to be locked.

If the *Lock flag* is 1, the corresponding columns are locked. This is the default and may be omitted. If the *Lock flag* is 0, the corresponding columns are unlocked

EXAMPLES

```

F.G.LockColumns 3      ALock 3rd column
F.G.LockColumns 3 0    AUnlock 3rd column
F.G.LockColumns (4 5)  ALock 4th & 5th cols
F.G.LockColumns (2 1p8 4) ALock 8 at 4
F.G.LockColumns 3      ALock 3rd column
F.G.LockColumns 3 0    AUnlock 3rd column
F.G.LockColumns (4 5)  ALock 4th & 5th cols
F.G.LockColumns (2 1p8 4) ALock 8 at 4

```

The result is an integer matrix containing the indices of all locked columns and the positions at which they are currently locked.

The expression:

```
F.G.LockColumns <@
```

may therefore be used to obtain the indices of the locked columns, and:

```
F.G.LockColumns(F.G.LockColumns <@) 0
```

unlocks all currently locked columns.

Locks are additive. If column 4 is locked, locking column 5 results in both columns 4 and 5 being locked.

A locked column remains fixed in position and does not scroll sideways. The user may enter and edit cells in a locked column in the normal way, but the behaviour of the various cell movement keys (Tab, left and right cursor, and so forth) differs when a locked column is encountered. As a general rule, if a keystroke attempts to move the cursor into a locked column from an adjacent column, and the adjacent column has been scrolled, it is unscrolled and the cursor remains in the (new) column adjacent to the fixed column. If not, the cursor moves into the locked column.

When you lock a column, the position you specify for it to be locked at is a position *in the data* and not the physical position of the column as displayed in the [Section 6.263](#). The physical column in the [Section 6.263](#) depends upon the value of the [Section 6.319](#) property at the time it was locked.

If *c* is the value specified for where a given column is to be locked, the value of the physical column *P* at which it will be displayed in the Grid named *GRID* is:

```
P←C-(2>GRID □WG 'Index')-□IO
```


Furthermore, the position of a locked column given by the result of the LockColumns method changes (with the [Section 6.319](#) property) as the [Section 6.263](#) is scrolled.

6.358 LockRows Method 226

Applies To: [Section 6.263](#)

Description

This method is used to lock one or more Rows of a [Section 6.263](#). However, LockRows is not supported in combination with hierarchical row titles as specified by the [Section 6.498](#) property.

The argument to LockRows is a 1 or 2-element vector as follows.

[1]	Row(s)	integer scalar, vector or matrix
[2]	Lock flag	0 or 1

Row(s) may be a scalar or a vector specifying the row or rows to be locked or unlocked. Alternatively, it may be a matrix whose first row specifies the data rows to be locked and whose second row specifies *where* in the Grid they are to be locked.

If the *Lock flag* is 1, the corresponding rows are locked. This is the default and may be omitted. If the *Lock flag* is 0, the corresponding rows are unlocked.

EXAMPLES

```
F.G.LockRows 3           aLock 3rd row
F.G.LockRows 3 0         aUnlock 3rd row
F.G.LockRows (4 5)       aLock 4th and 5th rows
F.G.LockRows (2 1p8 4)   aLock row 8 at 4
F.G.LockRows 3           aLock 3rd row
F.G.LockRows 3 0         aUnlock 3rd row
F.G.LockRows (4 5)       aLock 4th and 5th rows
F.G.LockRows (2 1p8 4)   aLock row 8 at 4
```

The result is an integer matrix containing the indices of all locked rows and the positions at which they are currently locked.

The expression:

```
F.G.LockRows <=0
```

may therefore be used to obtain the indices of the locked rows, and

```
F.G.LockRows(F.G.LockRows <⊖) 0
```

unlocks all currently locked rows.

Locks are additive. If row 4 is locked, locking row 5 results in both rows 4 and 5 being locked.

A locked row remains fixed in position and does not scroll vertically. The user may enter and edit cells in a locked row in the normal way, but the behaviour of the various cell movement keys (Tab, up and down cursor, and so forth) differs when a locked row is encountered. As a general rule, if a keystroke attempts to move the cursor into a locked row from an adjacent row, and the adjacent row has been scrolled, it is unscrolled and the cursor remains in the (new) row adjacent to the fixed row. If not, the cursor moves into the locked row.

When you lock a row, the position you specify for it to be locked at is a position *in the data* and not the physical position of the column as displayed in the Grid. The physical column in the Grid depends upon the value of the Index property at the time it was locked.

If R is the value specified for where a given row is to be locked, the value of the physical row P at which it will be displayed in the Grid named GRID is given by the expression:

```
P←R-(→GRID ⍴WG 'Index')-⍴IO
```

Furthermore, the position of a locked row given by the result of the LockRows method changes (with the Index property) as the Grid is scrolled.

6.359 LostFocus Event 41

Applies To

Description

If enabled, this event is generated when the user transfers the keyboard focus away from the object in question.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'LostFocus' or 41
[3]	Object name	character vector (name of object that has received the focus)

If the focus is transferred to a window that is not part of the Dyalog APL GUI Interface, the third element is an empty vector.

The LostFocus event is generated **after** the focus has changed. The default processing is therefore to take no action. However, if you inhibit the event by returning a 0 from your callback function, the focus is automatically restored to the object that had lost it.

6.360 LStyle Property

Applies To: [Section 6.94](#), [Section 6.190](#), [Section 6.355](#), [Section 6.450](#), [Section 6.472](#)

Description

This property determines the type of line used to draw a graphics object. It takes one of the following integer values, or, if the object contains more than one component, a vector of such values.

0	solid line
1	dashed line
2	dotted line
3	dash dotted line
4	dash dot dotted line
5	null line (invisible)

If LStyle is in the range 1-4, the gaps between the dashes and dots are drawn using the colour specified by [Section 6.45](#), or are left undrawn (that is, transparent) if [Section 6.45](#) is not defined.

In versions of Dyalog prior to v13.2 revision 19489, if LWidth specified a line width greater than 1 pixel, a solid line was drawn in the colour specified by the FCol Property, regardless of the value of [Section 6.360](#). From that revision onwards, if the value of LWidth is greater than 1 then the value of LStyle is honoured, but only the FCol of the line is honoured - the BCol is still ignored.

6.361 LWidth Property

Applies To: [Section 6.94](#), [Section 6.190](#), [Section 6.450](#), [Section 6.472](#)

Description

This property determines the width of line used to draw a graphics object. A positive value specifies the line width in pixels. A negative value specifies line width in units of the co-ordinate system defined for the object in the x direction. If the object contains more than one component, LWidth may be a vector.

In versions of Dyalog prior to v13.2 revision 19489, if LWidth specified a line width greater than 1 pixel, a solid line was drawn in the colour specified by the FCol Property, regardless of the value of [Section 6.360](#). From that revision onwards, if the value of LWidth is greater than 1 then the value of LStyle is honoured, but only the FCol of the line is honoured - the BCol is still ignored.

6.362 MakeGIF Method 261

Applies To: [Section 6.47](#)

Description

This method is used to generate a GIF representation of a picture from a Bitmap object suitable for display by a Web browser.

The MakeGIF method is niladic.

The result is an integer vector containing the encoded GIF image.

EXAMPLE

```
ρGIF←BM.MakeGIF
19620
```

6.363 MakePNG Method 260

Applies To: [Section 6.47](#)

Description

This method is used to generate a PNG (Portable Network Graphics) representation of a picture from a Bitmap object suitable for display by a Web browser.

The MakePNG method is niladic.

The result is an integer vector containing the encoded PNG image.

Example:

```
ρPNG←BM.MakePNG
4930
```

6.364 MapCols Property

Applies To: [Section 6.316](#)

Description

The MapCols property specifies whether or not the button colours in bitmaps and icons in an [Section 6.316](#) are re-mapped to reflect the users colour preferences. If your bitmaps and icons represent buttons using the standard windows button colours, this property causes those colours to be changed to suit the user's own colour scheme.

MapCols is a single number with the value 0 (no colour mapping) or 1 (colours are automatically re-mapped). The default is 0.

If MapCols is 1, the following colour mappings are performed:

Colour	Description	Mapped to
0 0 0	Black	Button Text
128 128 128	Dark grey	Button Shadow
191 191 191	Light grey	Button Face
192 192 192	Light grey	Button Face
255 255 255	White	Button Highlight

6.365 Marker Object

Purpose: A graphical object used to draw polymarkers.

Parents ActiveXControl Animation Bitmap Button ButtonEdit Combo ComboEx Edit Form Grid Group Label List ListView MDIClient Metafile Printer ProgressBar PropertyPage PropertySheet RichEdit Scroll Spinner Static StatusBar SubForm TabBar TipField ToolBar TrackBar TreeView UpDown

Children Timer

Properties Type Points Style Size FCol Coord Visible Event Dragable OnTop AutoConf
Data Accelerator KeepOnClose DrawMode MethodList ChildList EventList PropList

Methods Detach

Events Close Create DragDrop MouseDown MouseUp MouseMove MouseDbIClick Help
Select

Description

The [Section 6.449](#) property specifies one or more sets of points at which one or more sets of polymarkers are to be drawn.

The [Section 6.577](#) property determines the symbol that is drawn at each of a set of points. Marker styles are specified either by numbers which represent the following symbol shapes.

0	.
1	+
2	*
3	□
4	x
5	◇
6	◦

or by character vectors containing the names of [Section 6.47](#) or [Section 6.309](#) objects.

The height of each symbol is specified by the value of the [Section 6.556](#) property. However this applies only to [Section 6.577](#)s 1-6 and is ignored if [Section 6.577](#) is 0 or the name of a [Section 6.47](#). The colour of each symbol is specified by the [Section 6.205](#) property. The default is black.

The value of [Section 6.173](#) determines whether or not the object can be dragged. The value of [Section 6.35](#) determines whether or not the Marker object is resized when its parent is resized.

The structure of the property values is best considered separately for single and multiple sets of polymarkers.

Single Set of Polymarkers

For a single set of polymarkers, [Section 6.449](#) is either a 2-column matrix of (y,x) co-ordinates, or a 2-element vector of y and x co-ordinates respectively.

[Section 6.577](#) and [Section 6.556](#) are both simple scalar numbers.

[Section 6.205](#) is either a single number representing a standard colour, or a 3-element vector which specifies the marker colour explicitly in terms of RGB values.

EXAMPLES

First make a [Section 6.227](#):

```
'F' WC 'Form'
```

Draw a point at (y=20, x=10):

```
'F.M1' WC 'Marker' (20 10)
```

Draw a row of points at (y=20, x=10, 20, ... 90) : (Note scalar extension of y-coordinate)

```
'F.M1' WC 'Marker' (20(10×19))
```

Draw "+" symbols at each corner of a box:

```
Y ← 10 10 50 50
X ← 10 50 50 10
'F.M1' WC 'Marker' (Y X) 1
```

Ditto, but draw them 10% high:

```
'F.M1' WC 'Marker' (Y X) 1 10
```

Ditto, but use "*" symbols in green:

```
'F.M1' WC 'Marker' (Y X) 2 10 (0 255 0)
```

Multiple Sets of Polymarkers

To draw multiple sets of polymarkers with a single name, [Section 6.449](#) is a nested vector whose items are themselves 2-column matrices or 2-element nested vectors.

[Section 6.577](#) and [Section 6.556](#) may be simple scalars specifying a single type and/or size of symbol to be used for all the sets of polymarkers, or vectors specifying different symbols and/or sizes for each set.

[Section 6.205](#) may be a single number or a single (enclosed) 3-element vector applying to all the sets of polymarkers. Alternatively, [Section 6.205](#) may be a vector whose elements refer to each of the sets of polymarkers in turn. If so, the elements may be single numbers or nested RGB triplets, or a combination of the two.

EXAMPLES

First make a [Section 6.227](#):

```
'F' □WC 'Form'
```

Draw a "□" at (10,20) and a "◊" at (20,20):

```
'F.M1' □WC 'Marker' ((1 2p10 20)(1 2p20 20)) (3 5)
```

Draw "+" symbols at each corner of one box and "o" symbols at each corner of another:

```
Y1 X1 ← (10 10 50 50) (10 50 50 10)
Y2 X2 ← (20 20 40 40) (20 40 40 20)
'F.M1' □WC 'Marker' ((Y1 X1)(Y2 X2)) (1 6)
```

Ditto, but draw the "+" symbols with height 2% and the "o" symbols 5%:

```
'F.M1' □WC 'Marker' ((Y1 X1)(Y2 X2)) (1 6) (2 5)
```

Ditto, but draw the "+" symbols in red and the "o" symbols in blue:

```
'F.M1' □WC 'Marker' ((Y1 X1)(Y2 X2)) (1 6) (2 5)
('FCol' (255 0 0)(0 0 255))
```

6.366 Mask Property

Applies To: [Section 6.136](#), [Section 6.309](#)

Description

This property is used to specify how the bitmap for a [Section 6.136](#) or [Section 6.309](#) interacts with the pixels of the screen when it is displayed.

When a [Section 6.136](#) or [Section 6.309](#) is displayed, the colour of each pixel occupied by the object on the screen is determined by :

- The colour specified by [Section 6.48](#) via [Section 6.104](#)
- The value of Mask
- The existing colour of the screen pixel

Mask is a Boolean matrix with the same shape as the [Section 6.48](#) property. See [Section 6.136](#) and [Section 6.309](#) objects for further details.

6.367 MaskCol Property

Applies To: [Section 6.47](#), [Section 6.227](#)

Description

Specifies the transparent colour for a Bitmap or Form.

MaskCol may be an integer scalar or a 3-element integer vector.

If MaskCol is 0 (the default), no transparent colour is defined.

If MaskCol is a negative scalar, it specifies a standard Windows colour. See BCol for details.

Otherwise, MaskCol is a 3-element vector of integers in the range 0-255 that specifies the transparent colour in terms of RGB values (the intensity of the red, green and blue components of colour).

For a Bitmap, if MaskCol is non-zero, any pixels specified with the same colour will instead be displayed in whatever colour is underneath the Bitmap. This achieves similar behaviour to that of an Icon.

For a Form, if MaskCol is non-zero, any of the contents of the Form that are specified to be the same colour as MaskCol will be transparent. For example, if MaskCol is 255 0 0 (red), any red items contained in the Form will instead be transparent areas, displaying whatever is behind them on the screen. Mouse events generated over such transparent areas will be passed to any other windows behind them, and will not be reported on the Form itself.

6.368 Masked Property

Applies To: [Section 6.316](#)

Description

The Masked property specifies whether or not the [Section 6.316](#) will contain opaque or transparent images. It may be 0, 1(the default), 2, or 3.

Masked must be established when the [Section 6.316](#) is created by `WC` and may not subsequently be altered. An inappropriate value of Masked will cause the images to be drawn incorrectly.

If Masked is 0, the [Section 6.316](#) expects opaque [Section 6.47](#) objects.

If Masked is 1, the [Section 6.316](#) expects low-colour (4-bit or 8-bit) [Section 6.309](#) objects whose transparency is defined by their [Section 6.366](#) property.

If Masked is 2, the [Section 6.316](#) expects [Section 6.47](#) or [Section 6.309](#) objects whose alpha channel (the degree of transparency of each pixel) is encoded in its source file along with the colours.

If Masked is 3 and [Section 8.6](#)

is enabled, the behaviour is the same as if Masked were 2. If Native Look and Feel is not enabled, it behaves as if Masked were 1. This setting provides the greatest degree of portability for applications whose users may or may not have Native Look and Feel enabled. This value is used for the ImageLists on the Dialog Session CoolBars.

6.369 MaxButton Property

Applies To: [Section 6.227](#), [Section 6.307](#), [Section 6.578](#)

Description

This property determines whether or not an object has a "maximise" button. Pressing this button will cause a [Section 6.227](#) or [Section 6.307](#) to be resized to occupy the entire screen, or a [Section 6.578](#) to occupy the entire area of its parent. Pressing it again will restore the object to its original size. MaxButton is a single number with the value 0 (no maximise button) or 1 (maximise button is provided). The default is 1.

Note that MaxButton is independent of [Section 6.557](#), that is, you can define an object that can be maximised but not resized. If any of the properties MaxButton, [Section 6.388](#), [Section 6.580](#) and [Section 6.557](#) are set to 1, the object will have a title bar.

6.370 MaxDate Property

Applies To: [Section 6.58](#), [Section 6.143](#)

Description

The MaxDate property specifies the largest date that the user may select in a Calendar object or in the calendar drop-down of a [Section 6.143](#).

MaxDate is an [Section 8.5](#) value. Its default value is 11249470 which is the maximum date that the Calendar can display.

6.371 MaxLength Property

Applies To: [Section 6.56](#), [Section 6.185](#), [Section 6.563](#)

Description

This property specifies the maximum number of characters that the user may enter in a single-line [Section 6.185](#) object ([Section 6.577](#) 'Single') or in the edit field associated with a [Section 6.563](#). It does not apply to a multi-line [Section 6.185](#) object ([Section 6.577](#) 'Multi'). MaxLength does not limit the length of the vector that you may assign to the [Section 6.606](#) property using *Dyalog APL Language: Wc* or *Dyalog APL Language: Ws*. However, if you overfill the field in this way, the user must delete excess characters before the object will accept further input.

6.372 MaxSelCount Property

Applies To: [Section 6.58](#)

Description

The MaxSelCount property specifies the maximum number of contiguous days that the user may select in a [Section 6.58](#) object.

MaxSelCount is an integer whose default value is 7.

MaxSelCount is ignored unless the [Section 6.577](#) property of the [Section 6.58](#) object is set to 'Multi'.

6.373 MDIActivate Event 42

Applies To: [Section 6.578](#)

Description

This event is generated when the user activates a particular [Section 6.578](#) that is the child of an [Section 6.378](#). This occurs when the user clicks the left mouse button in the [Section 6.578](#) or selects it from the menu nominated for this purpose (see [Section 6.380](#) property). You may also call MDIActivate as a method.

Note that this event is reported after the action has taken place and cannot be disabled by returning 0 from a callback function or by setting its action code to -1.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'MDIActivate' or 42
[3]	Object name	character vector

Note that the 3rd element of the event message is either an empty vector or contains the name of the [Section 6.578](#) that was previously the active one in the same [Section 6.378](#).

6.374 MDIActive Property

Applies To: [Section 6.378](#)

Description

This property contains the name of the [Section 6.578](#) owned by the [Section 6.378](#) that is currently active. Only one [Section 6.578](#) may be active at a time. You can switch between [Section 6.578](#)s in an MDI application under program control by setting this property with *Dyalog APL Language: Ws*. You can also do this by generating an [Section 6.373](#) event.

See also [Section 6.375](#) property.

6.375 MDIActiveObject Property

Applies To: [Section 6.378](#)

Description

This property contains a ref to the [Section 6.578](#) owned by the [Section 6.378](#) that is currently active.

See also [Section 6.374](#) property.

6.376 MDIArrange Method 112

Applies To: [Section 6.378](#)

Description

This method causes the [Section 6.378](#) object to organise the icons associated with any minimised child [Section 6.227s](#) into regimented rows and columns. To permit the user to carry out this action, it is recommended that a suitable callback function or expression is attached to a [Section 6.384](#) or [Section 6.55](#). The callback function or expression should then call MDIArrange.

The MDIArrange method is niladic.

6.377 MDICascade Method 110

Applies To: [Section 6.378](#)

Description

This method causes the [Section 6.378](#) object to organise its child [Section 6.227s](#) in an overlapping fashion. To permit the user to carry out this action, it is recommended that a suitable callback function or expression is attached to a [Section 6.384](#) or [Section 6.55](#). The callback function or expression should then call MDICascade.

The MDICascade method is niladic.

6.378 MDIClient Object

Purpose: Implements Multiple Document Interface (MDI) behaviour.

Parents ActiveXControl Form SubForm

Children Circle Ellipse Font Marker Poly Rect SubForm Text Timer

Properties Type Posn Size Coord Border Event BCol Picture IconObj CursorObj YRange XRange Data Attach EdgeStyle Handle MDIActive MDIActiveObject Hint HintObj Tip TipObj Translate Accelerator KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach MDICascade MDITile MDIArrange GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap Select

Description

The multiple-document interface (MDI) is a document-oriented interface that is commonly used by word-processors, spreadsheets and other applications that deal with *documents*. An MDI application allows the user to display multiple documents at the same time, with each document displayed in its own window.

The MDIClient object is a container object that effectively specifies the client area within the parent [Section 6.227](#) in which the [Section 6.578](#) are displayed. The MDIClient object also imposes special MDI behaviour which is quite different from that where a [Section 6.578](#) is simply the child of another [Section 6.227](#).

By default, the MDIClient occupies the entire client area within its parent [Section 6.227](#). This is the area within the [Section 6.227](#) that is not occupied by [Section 6.127s](#), [Section 6.383s](#), [Section 6.621s](#), [Section 6.624s](#), [Section 6.582s](#), [Section 6.585](#) and [Section 6.574s](#). In most applications it is therefore not necessary to specify the position and size of the MDIClient object, although you may do so if you want to reserve additional space in the parent [Section 6.227](#) for other objects.

Each of the four sides of an MDIClient object is automatically *attached* to the corresponding side of its parent [Section 6.227](#) and maintains its position when the parent [Section 6.227](#) is resized. This means that a default MDIClient always occupies the entire client area of its parent [Section 6.227](#), regardless of how the parent is resized.

The appearance of the MDIClient may be changed using its [Section 6.49](#), [Section 6.45](#) and [Section 6.447](#) properties. The [Section 6.184](#) property has no direct effect and is provided only to pass on a value to its child [Section 6.227s](#).

The [Section 6.374](#) and [Section 6.375](#) properties contain the name of and a ref to the [Section 6.578](#) that currently has the focus. You may set these properties as well as query them.

You can call methods which cause the MDIClient to organise its child [Section 6.578s](#) in some way. These methods are as follows:

MDICascade	Causes the MDIClient to organise its child Forms in an overlapping manner.
MDITile	Causes the MDIClient to arrange its child Forms as a row or column.
MDIArrange	Causes the MDIClient to arrange the icons associated with any minimised child Forms in an orderly fashion.

6.379 MDIDeactivate Event 43

Applies To: [Section 6.578](#)

Description

This event is generated when the user activates a different [Section 6.578](#) that is the child of an [Section 6.378](#), thereby de-activating the current one which causes this event. This occurs when the user clicks the left mouse button in another [Section 6.578](#) or selects it from the menu nominated for this purpose (see [Section 6.380](#) property). You may also call MDIDeactivate as a method.

Note that this event is reported after the action has taken place and cannot be disabled by returning 0 from a callback function or by setting its action code to -1.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'MDIDeactivate' or 43
[3]	Object name	character vector

Note that the 3rd element of the event message contains the name of the [Section 6.578](#) that has now been made the active one in the same [Section 6.378](#).

6.380 MDIMenu Property

Applies To: [Section 6.383](#)

Description

This property specifies the name of, or ref to, the [Section 6.382](#) object that is nominated as the "Window" menu in an MDI application. If such a menu is defined, the [Section 6.66s](#) of all the child [Section 6.227s](#) are automatically added to it below any other [Section 6.382](#) or [Section 6.384](#) objects that the application has created directly. This list is separated from the preceding items by a separator. The entry for the currently active [Section 6.578](#) is checked and the user may switch between [Section 6.578s](#) by selecting from this list.

Note that the additional separator and the items representing the list of child forms are not Dyalog APL/W objects and may not be accessed by the application. If you prefer to maintain your own "window list" you should not use this property.

6.381 MDITile Method 111

Applies To: [Section 6.378](#)

Description

This method causes the [Section 6.378](#) object to organise its child [Section 6.227s](#) as a row or column. To permit the user to carry out this action, it is recommended that a suitable callback function or expression is attached to a [Section 6.384](#) or [Section 6.55](#). The callback function or expression should then call the MDITile method.

Note that because there are restrictions concerning the minimum height and width of a window, MS-Windows does not necessarily respond as requested. If the [Section 6.378](#) is itself of insufficient size, or if it contains a large number of child [Section 6.227s](#), Windows may choose to tile the [Section 6.227s](#) in a row when a column was specified or vice versa. It may also choose to ignore the event entirely.

The argument to MDITile is 0, or a single item as follows:

[1]	Tile Mode	0 (vertical) 1 (horizontal)

If the argument is $\emptyset$, the *Tile Mode* defaults to 0.

6.382 Menu Object

Purpose: This is a pop-up object which allows the user to initiate an action or to select an option using a "menu".

Parents ActiveXControl Calendar CoolBand CoolBar DateTimePicker Form Grid Menu MenuBar OLEServer Root StatusField SubForm SysTrayItem TCPSocket ToolBar ToolControl

Children Bitmap Menu MenuItem Separator Timer

Properties Type Caption Posn Coord Align Active Event FontObj FCol BCol BtnPix Data EdgeStyle Handle Translate Accelerator KeepOnClose ImageListObj ImageIndex MethodList ChildList EventList PropList

Methods Detach Wait

Events Close Create DropDown Select

Description

For a Menu that is owned by a [Section 6.383](#) or another Menu, the [Section 6.66](#) property determines the text string that is displayed as the "choice". The Menu is then popped up by the user clicking on this text. It is automatically popped down when the user chooses an option (by selecting a [Section 6.384](#)) or cancels the operation (by clicking elsewhere).

If a Menu belongs to a [Section 6.227](#), [Section 6.578](#) or is a top-level object, it must be popped up by the application. This is commonly done in response to a [Section 6.392](#) event. A Menu is popped-up by calling *Dyalog APL Language: Dq* with only the name of the Menu as its argument. The user may therefore not interact with any other object until a selection is made or until the operation is cancelled. When either occurs, the Menu is automatically popped down and de-activated, and its *Dyalog APL Language: Dq* terminates.

The Menu object does not have a [Section 6.556](#) property. Instead, its size is determined automatically by its contents.

If a Menu is owned by a [Section 6.383](#) or by another Menu, its position within its parent is also calculated automatically, dependent on the order in which other related objects are established. The [Section 6.452](#) property may however be used to **insert** a

new Menu into an existing structure. For example, having defined three Menu objects as children of a [Section 6.383](#), you can insert a fourth one between the first and the second by specifying its [Section 6.452](#) to be 2. Note that the value of [Section 6.452](#) for the Menus that were previously second and third will then be reset to 3 and 4 respectively.

If a Menu is a child of a [Section 6.383](#) which is itself a child of a [Section 6.227](#) or [Section 6.578](#), the Align property can be set to 'Right'. This is used to position a single Menu (or [Section 6.384](#)) at the rightmost end of a [Section 6.383](#). This does not apply if the [Section 6.383](#) is owned by a [Section 6.624](#).

The [Section 6.53](#) property is used to display a picture in a Menu. [Section 6.53](#) specifies the names of, or refs to, three [Section 6.47](#) objects. The first [Section 6.47](#) is displayed when the Menu does not have the focus (normal), the second when it does have the focus (highlighted). The third [Section 6.47](#) is displayed when the Menu is made inactive ([Section 6.5](#) property is 0). If [Section 6.66](#) is also defined, it is displayed **on top of** the bitmaps.

If the Menu is a submenu (owned by a Menu), you may set its [Section 6.184](#) property to 'Plinth'. This causes the Menu to take on an appearance that is similar to a pushbutton and be raised when not selected and recessed when selected. Note that to enable 3-dimensional appearance, you must set [Section 6.184](#) to something other than 'None' for all the objects above the Menu in the tree.

[Section 6.184](#), [Section 6.53](#), [Section 6.222](#), [Section 6.205](#) and [Section 6.45](#) do not affect the appearance of a Menu if it is the direct child of a [Section 6.383](#). However, the [Section 6.184](#) property **must** be set to something other than 'None' if you want its children Menu and [Section 6.384](#) objects to have a 3-dimensional appearance.

6.383 MenuBar Object

Purpose: Specifies a horizontal menu bar displayed at the top of a [Section 6.227](#).

Parents ActiveXControl Form SubForm ToolControl

Children Bitmap Menu MenuItem Separator Timer

Properties Type Visible Event FontObj Data EdgeStyle MDIMenu Handle Translate KeepOnClose MethodList ChildList EventList PropList

Methods Detach

Events Close Create

Description

Unless it is made invisible the MenuBar is always available to the user to initiate actions or to select options. A MenuBar has a fixed position and size.

It is possible to have more than one MenuBar associated with the same [Section 6.227](#) or [Section 6.578](#), but only one of them should be [Section 6.645](#) at any one time.

The following example illustrates how a menu structure can be built up from a MenuBar. For clarity, the example is indented, and the definition of the [Section 6.196](#) property is omitted.

```
'F'           □WC 'Form' 'Menu Example'
'F.M'         □WC 'MenuBar'
'F.M.FILE'    □WC 'Menu' '&File'
'F.M.MAT'     □WC 'Menu' '&Materials'
'F.M.MAT.B'   □WC 'MenuItem' '&Brick'
'F.M.MAT.C'   □WC 'MenuItem' '&Concrete'
'F.M.MAT.S'   □WC 'MenuItem' '&Stone'
'F.M.MAT.SEP' □WC 'Separator'
'F.M.MAT.W'   □WC 'Menu'      '&Wood'
'F.M.MAT.W.O' □WC 'MenuItem' '&Oak'
'F.M.MAT.W.T' □WC 'MenuItem' '&Teak'
'F.M.MAT.W.M' □WC 'MenuItem' '&Mahogany'
```

Note that putting a [Section 6.522](#) (either [Section 6.577](#)) in a MenuBar has the effect of breaking the bar vertically, that is, the next [Section 6.382](#) or [Section 6.384](#) you add will appear on the left-hand side on the line below.

The [Section 6.184](#) property has no effect on the appearance of a MenuBar or of a direct child of a MenuBar. However, if you want the sub-menus to have a 3-dimensional appearance, you **must** set the [Section 6.184](#) property of the MenuBar to something other than 'None'.

If the MenuBar is owned by a [Section 6.227](#) that is the parent of an [Section 6.378](#), you can set the [Section 6.380](#) property to the name of the [Section 6.382](#) you wish to nominate as the window menu. This menu will automatically be updated with the [Section 6.66s](#) of the child [Section 6.578](#) and may be used to select the currently active one.

6.384 MenuItem Object

Purpose: This object allows the user to initiate an action or to select an option from a menu.

Parents Menu MenuBar

Children Bitmap Timer

Properties Type Caption Posn Style Align Active Event Checked FontObj FCol BCol BtnPix Data EdgeStyle Hint HintObj Tip TipObj Translate Accelerator KeepOnClose ImageIndex MethodList ChildList EventList PropList

Methods Detach

Events Close Create Select

Description

The [Section 6.66](#) property determines the text string that is displayed in its parent as the menu option. The size of a MenuItem is determined by the size of its [Section 6.66](#), or by the size of the largest object ([Section 6.382](#), MenuItem or [Section 6.522](#)) with the same parent. The position of the MenuItem is normally determined by the order in which it is created in relation to other objects with the same parent. However, you can use the [Section 6.452](#) property to **insert** a new MenuItem into an existing structure. For example, having defined three MenuItem objects as children of a [Section 6.382](#), you can insert a fourth one between the first and the second by specifying its [Section 6.452](#) to be 2. Note that the value of [Section 6.452](#) for the MenuItems that were previously second and third will then be reset to 3 and 4 respectively.

The [Section 6.577](#) property may be 'Check' (the default) or 'Radio'. [Section 6.577](#) determines the type of graphic displayed alongside the Caption if the MenuItem is *checked*.

The [Section 6.90](#) property is a single number with the value 0 or 1. 0 means not checked (the default), 1 means checked and a tick mark or radio dot is placed alongside its [Section 6.66](#). This property is frequently used to indicate which of a choice of options is currently set.

If a MenuItem is a child of a [Section 6.383](#) which is itself a child of a [Section 6.227](#) or [Section 6.578](#), the Align property can be set to 'Right'. This is used to position a single MenuItem (or [Section 6.382](#)) at the rightmost end of a [Section 6.383](#). This does not apply if the [Section 6.383](#) is owned by a [Section 6.624](#).

If you set the [Section 6.184](#) property to 'Plinth', the MenuItem will take on an appearance that is similar to a pushbutton and be raised when not selected and recessed when selected. Note that to enable 3-dimensional appearance, you must set [Section 6.184](#) to something other than 'None' for all the objects above the MenuItem in the tree.

The [Section 6.53](#) property is used to display a picture in a MenuItem. [Section 6.53](#) specifies the names of, or refs to, three [Section 6.47](#) objects. The first [Section 6.47](#) is displayed when the MenuItem does not have the focus (normal), the second when it does have the focus (highlighted). The third [Section 6.47](#) is displayed when the MenuItem is made inactive ([Section 6.5](#) property is 0). If [Section 6.66](#) is also defined, it is displayed **on top of** the bitmaps.

Alternatively, you may display an image alongside the [Section 6.66](#) using the [Section 6.315](#) property. This selects a picture from the [Section 6.316](#) associated with the [Section 6.317](#) property of the parent [Section 6.382](#).

[Section 6.184](#), [Section 6.53](#), [Section 6.225](#), [Section 6.205](#) and [Section 6.45](#) are not effective if the MenuItem is the direct child of a [Section 6.383](#).

A MenuItem generates a [Section 6.513](#) event (if enabled) when the user chooses it.

6.385 Metafile Object

Purpose: This object represents a picture in Windows Metafile format.

Parents ActiveXControl Bitmap CoolBand Form Group OLEServer Printer PropertyPage PropertySheet Root Static SubForm TCPSocket ToolBar ToolControl

Children Circle Ellipse Font Image Marker Poly Rect Text Timer

Properties Type File Size Coord RealSize Event YRange XRange Data Handle Translate Accelerator KeepOnClose MethodList ChildList EventList PropList

Methods Detach FileRead FileWrite

Events Close Create Select

Description

The Windows Metafile is a mechanism for representing a picture in terms of a collection of graphical components. Windows Metafiles are distributed in special files (.WMF) from which they are loaded into memory for use by an application. Once

loaded a Metafile is a Windows "resource" that can be used in a variety of ways. The Metafile object represents this resource.

The [Section 6.207](#) property specifies the name of a .WMF file from which the Metafile is to be loaded or to which it is to be saved. If you specify [Section 6.207](#) with *Dyalog APL Language: Wc* the Metafile object is loaded from it. If you specify [Section 6.207](#) with *Dyalog APL Language: Ws* no action takes place until you instruct the Metafile object to re-initialise itself from the file or to save itself to the file. These operations are performed using the [Section 6.212](#) and [Section 6.213](#) methods. If you omit the [Section 6.207](#) property in the argument to *Dyalog APL Language: Wc* or if you specify a null vector, the Metafile object is initially empty. The following example loads the picture defined by the GOLF.WMF Metafile that is distributed with Microsoft Office.

```
'GOLF' ⍴WC 'Metafile' 'C:\MSOFFICE\CLIPART\GOLF'
```

Whether or not the Metafile object is initialised from a file, you can add graphical components to it by creating child objects. However the Metafile behaves like a [Section 6.47](#) object in that its children cannot be modified using *Dyalog APL Language: Ws* nor can they be removed using `⍴EX`. The components of a Metafile that has been initialised from a .WMF file also cannot be referenced in any way. It is therefore recommended that you use unnamed objects when you create the graphical components of a Metafile.

The following statements create an empty Metafile called MF and then draw a line and circle in it.

```
'MF' ⍴WC 'Metafile'
'MF.' ⍴WC 'Poly' (50(10 90))
'MF.' ⍴WC 'Circle' (50 50) 30
```

Like the [Section 6.47](#), [Section 6.309](#), [Section 6.222](#) and [Section 6.136](#) objects, the Metafile is a resource that is not visible until it is used. This is done by setting the [Section 6.447](#) property of another object ([Section 6.227](#), [Section 6.313](#), [Section 6.573](#) or [Section 6.578](#)) to the name of, or ref to, the Metafile object. For example, to display the Metafile MF in a [Section 6.227](#), you could type :

```
'TEST' ⍴WC 'FORM' ('Picture' 'MF')
```

You can also copy a Metafile object to the Windows Clipboard from where it can be pasted into another application. This is done by creating a [Section 6.99](#) object and then setting its [Section 6.386](#) property to the name of the Metafile object to be exported. For example :

```
'CL' □WC 'Clipboard'
'CL' □WS 'MetafileObj' 'MF'
```

The [Section 6.213](#) method may be used to save a Metafile object on a file. The following statements save the Metafile MF in a file called TEST.WMF.

```
MF.File←'TEST'
MF.FileWrite
```

The [Section 6.556](#) property determines the granularity of the Metafile. Its default value is the size of its parent. If you intend to replay the Metafile at higher resolution, you should set [Section 6.556](#) accordingly.

The [Section 6.471](#) property specifies the suggested size of a Metafile in units of 0.01mm. Setting [Section 6.471](#) has the effect of making the Metafile *placeable*. Certain programs (such as Word for Windows) only support placeable metafiles.

6.386 MetafileObj Property

Applies To: [Section 6.99](#)

Description

This property is used to copy graphical data to and from the Windows clipboard using the Windows Metafile format.

When you set the MetafileObj property of a [Section 6.99](#) object to the name of the [Section 6.385](#) object using *Dyalog APL Language: Ws* its contents are copied to the Windows clipboard in Windows Metafile format.

To import a picture that has been stored in the Windows clipboard in Metafile format you use *Dyalog APL Language: Wg*. This returns a nested array whose elements correspond to the graphical components of the picture. Each of the elements of the array may be used as the arguments of *Dyalog APL Language: Wc* to draw the corresponding component of the picture. For example, if the picture stored in C:\MSOFFICE\CLIPART\BIRD.WMF is copied to the Windows clipboard, it may be imported into Dyalog APL/W as follows :

```
BIRD ← 'CL' □WG 'MetafileObj'
ρBIRD
```

4

Each of the items in BIRD is a 2-element vector. The first element is a "dummy" object name which you may use or ignore as you wish. The second element is an array that

defines a graphical object and is suitable as the right argument of *Dyalog APL Language: Wc*. For example :

```

      2>4>BIRD
POLY 191 397  FSTYLE 0  FILLCOL 0 0 0  ...

190 402
187 406
182 409
176 410
172 409
168 406
165 402
164 397
165 391
168 387
172 384
176 383
182 384
187 387
190 391
191 397
189 395
191 397

```

From this array, you can rebuild the imported picture component by component, either as a [Section 6.385](#) object or directly onto a [Section 6.227](#), [Section 6.573](#) or another object. The following example draws the picture in a [Section 6.227](#) using the dummy names supplied.

```

'TEST' □WC 'FORM' ('Coord' 'User')
'TEST' □WS ('YRange' 0 1024)('XRange' 0 2048)
TEST.□WC/'BIRD

```

Notice that the co-ordinates of each of the graphical components are typically integers in a co-ordinate system that extends from 0 to 1024 in the y-direction and 0 to 2048 in the x-direction. The simplest way to draw the picture is therefore to set up the same co-ordinate system on a [Section 6.227](#) as in the example above.

6.387 MethodList Property

Applies To

Description

This property reports the names of all the methods supported by a particular COM object or instance of an OLE control. It is a vector of character vectors returned by `WC`. It may not be set using `WC` or `WS`.

6.388 MinButton Property

Applies To: [Section 6.227](#), [Section 6.307](#), [Section 6.578](#)

Description

This property determines whether or not an object has a "minimise" button. Pressing this button will cause the object to be iconified. Pressing it again will restore the object to its original size. MinButton is a single number with the value 0 (no minimise button) or 1 (minimise button is provided). The default is 1.

Note that MinButton is independent of [Section 6.557](#), that is, you can define an object that can be minimised but not resized.

If any of the properties MinButton, [Section 6.369](#), [Section 6.580](#), and [Section 6.398](#) are set to 1, the object will have a title bar.

6.389 MinDate Property

Applies To: [Section 6.58](#), [Section 6.143](#)

Description

The MinDate property specifies the smallest date that the user may select in a [Section 6.58](#) or [Section 6.143](#) object.

MinDate is an [Section 8.5](#) value. Its default value is -109206 which is the minimum date that the Calendar can display.

6.390 MonthDelta Property

Applies To: [Section 6.58](#), [Section 6.143](#)

Description

The MonthDelta property specifies the number of months by which a [Section 6.58](#) object scrolls when the user clicks its scroll buttons.

MonthDelta is an integer or an empty vector (zilde). The latter means that the [Section 6.58](#) object scrolls by the number of months that are currently displayed in its window. This is the default.

6.391 MouseButtonClick Event 5

Applies To

Description

If enabled, this event is reported when the user presses and then releases a mouse button twice within a short space of time. The duration of this time is set through the Windows Control Panel.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'MouseButtonClick' or 5
[3]	Y	y-position of mouse (number)
[4]	X	x-position of mouse (number)
[5]	Button	button double clicked (number) 1 = left button 2 = right button 4 = middle button
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down

In a graphical object ([Section 6.94](#), [Section 6.190](#), [Section 6.313](#), [Section 6.365](#), [Section 6.450](#) and [Section 6.472](#)), the position of the mouse is reported relative to the top-left corner of its bounding rectangle.

If you enable [Section 6.392](#) and [Section 6.396](#) events in addition to MouseButtonClick events, double-clicking a mouse button will generate the following sequence of events :

1. [Section 6.392](#)
2. [Section 6.396](#)
3. MouseButtonClick
4. [Section 6.396](#)

6.392 MouseDown Event 1

Applies To

Description

If enabled, this event is reported when the user presses one of the mouse buttons. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'MouseDown' or 1
[3]	Y	y-position of mouse (number)
[4]	X	x-position of mouse (number)
[5]	Button	button pressed (number) 1 = left button 2 = right button 4 = middle button
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down

If you enable this event it is advisable that you ALSO enable [Section 6.396](#) events. Otherwise, the slight delay in running your callback function will cause the down and up sequence to be **reversed**.

In a graphical object ([Section 6.94](#), [Section 6.190](#), [Section 6.313](#), [Section 6.365](#), [Section 6.450](#) and [Section 6.472](#)), the position of the mouse is reported relative to the top-left corner of its bounding rectangle.

6.393 MouseEnter Event 6

Applies To

Description

If enabled, this event is reported when the user moves the mouse pointer into (over) an object. The event message reported as the result of *Dyalog APL Language: Dq*, or

supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'MouseEnter' or 6
[3]	Object name	character vector (name of previous object)

This event is generated when the user moves the mouse pointer across the boundary and into an object. The first element of the event message is the name of the object over which the mouse pointer now resides. The 3rd element of the event message contains the name of the object that was previously under the mouse pointer, or is an empty vector if the mouse pointer was not previously over a Dyalog APL/W object.

6.394 MouseLeave Event 7

Applies To

Description

If enabled, this event is reported when the user moves the mouse pointer out of an object. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'MouseLeave' or 7
[3]	Object name	character vector (name of new object)

This event is generated when the user moves the mouse pointer across the boundary and away from an object. The first element of the event message contains the name of the object that previously contained the mouse pointer and which generated the event when it crossed its boundary. The third element contains the name of the object which now contains the mouse pointer or is an empty vector if the mouse pointer is not now over a Dyalog APL/W object.

6.395 MouseMove Event 3

Applies To

Description

If enabled, this event is reported when the user moves the mouse. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'MouseMove' or 3
[3]	Y	y-position of mouse (number)
[4]	X	x-position of mouse (number)
[5]	Button	button released (number) 1 = left button 2 = right button 4 = middle button
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down

In a graphical object ([Section 6.94](#), [Section 6.190](#), [Section 6.313](#), [Section 6.365](#), [Section 6.450](#) and [Section 6.472](#)), the position of the mouse is reported relative to the top-left corner of its bounding rectangle.

Note that rapid movement of the mouse will not necessarily cause an overwhelming number of MouseMove events to be reported, as several small movements are automatically combined into one large one.

6.396 MouseUp Event 2

Applies To

Description

If enabled, this event is reported when the user releases one of the mouse buttons. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'MouseUp' or 2
[3]	Y	y-position of mouse (number)
[4]	X	x-position of mouse (number)
[5]	Button	button released (number) 1 = left button 2 = right button 4 = middle button
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down

In a graphical object ([Section 6.94](#), [Section 6.190](#), [Section 6.313](#), [Section 6.365](#), [Section 6.450](#) and [Section 6.472](#)), the position of the mouse is reported relative to the top-left corner of its bounding rectangle.

6.397 MouseWheel Event 8

Applies To

Description

If enabled, this event is reported when the user rotates the mouse wheel.

The event message reported as the result of `get`, or supplied as the right argument to your callback function, is a 9-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'MouseWheel ' or 8
[3]	Y	y-position of mouse (number)
[4]	X	x-position of mouse (number)
[5]	Button	button pressed 1 = left button 2 = right button 4 = middle button
[6]	Shift State	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down
[7]	Delta	integer
[8]	Lines	integer
[9]	Wheel Delta	integer

The value of *Delta* indicates the distance that the wheel is rotated, expressed in multiples or divisions of *Wheel Delta*. A positive value indicates that the wheel was rotated forward, away from the user; a negative value indicates that the wheel was rotated backward, toward the user.

Lines specifies the number of lines to scroll when the wheel is rotated by 1 *Mouse Delta* unit. A value of -1 indicates that a whole screen is to be scrolled. These values are defined by the user's preferences (*Control Panel/Mouse*).

6.398 Moveable Property

Applies To: [Section 6.227](#), [Section 6.307](#), [Section 6.578](#)

Description

This property determines whether or not an object can be moved by the user. It is a single number with the value 0 (object cannot be moved) or 1 (object is moveable). If any of the properties [Section 6.388](#), [Section 6.369](#), [Section 6.580](#), and *Moveable* are set to 1, the object will have a title bar. However, an object with a title bar is not necessarily moveable.

6.399 MsgBox Object

Purpose: Provides a "modal" dialog box for displaying messages, errors, warnings and other information. The dialog box has a title, one or more lines of text, and up to three buttons.

Parents ActiveXControl Calendar CoolBand DateTimePicker Form Grid OLEServer PropertyPage PropertySheet Root SubForm TCPSocket ToolBar ToolControl

Children Timer

Properties Type Caption Text Style Btns Default Event Data EdgeStyle KeepOnClose MethodList ChildList EventList PropList

Methods Detach Wait

Events Close Create MsgBtn1 MsgBtn2 MsgBtn3

Description

The [Section 6.66](#) property determines the text displayed in the object's title bar.

The [Section 6.606](#) property determines the text to be displayed as the message.

The [Section 6.577](#) property determines the type of icon which is displayed. This is a character vector with one of the following values:

'Msg '	no icon (the default)
'Info '	information message icon
'Query '	query (question) icon
'Warn '	warning icon
'Error '	critical error icon

The [Section 6.54](#) property determines the set of buttons to be displayed. It is a simple vector (one button) or a matrix with up to 3 rows, or a vector of up to 3 character vectors specifying the captions for up to 3 buttons. MS-Windows restricts you to a fixed set of button captions which are described below. However, the property has been designed more generally to be useful under different GUIs and perhaps later revisions of Windows. The buttons are arranged along the bottom of the dialog box in the order specified.

The [Section 6.54](#) property may specify one of six sets of buttons as follows:

- 'OK '
- 'OK ' 'CANCEL '
- 'RETRY' 'CANCEL '
- 'YES' 'NO '
- 'YES' 'NO' 'CANCEL '
- 'ABORT' 'RETRY' 'IGNORE '

If any other combination is specified, *Dyalog APL Language: Wc* and *Dyalog APL Language: Ws* will report a DOMAIN ERROR. The names of the buttons are however case-insensitive, so the system will accept 'ok', 'Ok', 'oK' or 'OK'. If [Section 6.54](#) is not specified, it assumes a default according to [Section 6.577](#) as follows:

Style	Btns
'Msg' or 'Info'	'OK '
'Warn' or 'Error'	'OK ' 'CANCEL '
'Query'	'YES' 'NO '

The [Section 6.148](#) property may be used to determine which of the buttons is the "default" button, that is, the one which initially has the focus and is "selected" when the user presses the Enter key. It has the value 1, 2 or 3. If [Section 6.148](#) is not specified, the first button is the "default" button. Note that if the user switches focus to another button and presses Enter, this action selects the button with the focus.

Like a pop-up (floating) [Section 6.382](#), the MsgBox object is unusual in that it is strictly modal. It is created by *Dyalog APL Language: Wc* in the normal way, but at that stage is invisible and inactive. It is activated ONLY when *Dyalog APL Language: Dq* is called with the name of the MsgBox as the argument. When this is done, the MsgBox object pops up and is activated. Because there is no other object specified in the argument to *Dyalog APL Language: Dq*, all other objects are de-activated. The only thing that the user can do (within the APL application) is to press one of the buttons in the MsgBox. When this happens, the MsgBox automatically pops down, the callback function (if any) is fired, and then *Dyalog APL Language: Dq* terminates.

Notice that the position and size of the MsgBox are determined by Windows and are fixed, although the MsgBox may be moved by the user after it has been displayed.

The MsgBox object generates one of three events; [Section 6.400](#) (61), [Section 6.401](#) (62), or [Section 6.402](#) (63) depending upon which button is pressed.

```

Caption←'Default MsgBox' ♦ Text←'Hello World'
'Msg' □WC 'MsgBox' Caption Text ♦ □DQ 'Msg'

Caption←'Information MsgBox' ♦ Text←'Update Completed'
'Msg' □WC 'MsgBox' Caption Text 'Info' ♦ □DQ 'Msg'

Caption←'Query MsgBox' ♦ Text←'Save Changes'
'Msg' □WC 'MsgBox' Caption Text 'Query' ♦ □DQ 'Msg'

Caption←'Warning MsgBox'
Text←'Calculations will take 10 minutes'
'Msg' □WC 'MsgBox' Caption Text 'Warn' ♦ □DQ 'Msg'

Caption←'Error MsgBox'
Text←'Data out of range'
'Msg' □WC 'MsgBox' Caption Text 'Error' ♦ □DQ 'Msg'

Caption←'Custom MsgBox'
Text←'You can have a multi-line'
Text,←'message if you want one'
B←'ABORT' 'RETRY' 'IGNORE'
'Msg' □WC 'MsgBox' Caption Text 'Info' B ♦ □DQ 'Msg'

```

6.400 MsgBtn1 Event 61

Applies To: [Section 6.399](#)

Description

If enabled, this event is reported when the user responds to a [Section 6.399](#) object by clicking its first (leftmost) button. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'MsgBtn1' or 61

6.401 MsgBtn2 Event 62

Applies To: [Section 6.399](#)

Description

If enabled, this event is reported when the user responds to a [Section 6.399](#) object by clicking its second (from the left) button. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'MsgBtn2' or 62

6.402 MsgBtn3 Event 63

Applies To: [Section 6.399](#)

Description

If enabled, this event is reported when the user responds to a [Section 6.399](#) object by clicking its third (from the left) button. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'MsgBtn3' or 63

6.403 MultiColumn Property

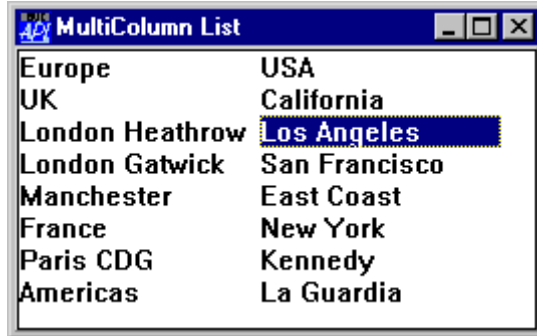
Applies To: [Section 6.346](#)

Description

MultiColumn is Boolean and specifies whether or not a List object displays its items in a single column (0, the default) or in multiple columns (1). MultiColumn may only be set by `⌈WC` and cannot be changed using `⌈WS` after the object has been created. Note that a MultiColumn List will use the minimum number of columns that are required to make the items fit within it and will reconfigure itself automatically when resized.

The following example illustrates its use.

```
'F'WC'Form' 'MultiColumn List'('Size' 23 32)
'F.L'WC'LIST' AIRPORTS (0 0)(100 100)('MultiColumn' 1)
```



6.404 MultiLine Property

Applies To: [Section 6.585](#), [Section 6.624](#)

Description

The MultiLine property determines whether or not the tabs or buttons will be arranged in multiple flights or multiple rows/columns in a [Section 6.585](#) or [Section 6.624](#) object.

MultiLine is a single number with the value 0 (single flight of tabs, or single row/column of buttons) or 1 (multiple flights of tabs or multiple rows/columns of buttons); the default is 0.

If MultiLine is 0 and there are more tabs or buttons than will fit in the space provided, the [Section 6.585](#) displays an UpDown which allows the user to scroll.

However, If MultiLine is 0 in a [Section 6.624](#), the buttons are clipped, and the user may have to resize the object to see them all.

See also: [Section 6.336](#), [Section 6.590](#).

6.405 MultiSelect Property

Applies To: [Section 6.585](#)

Description

The [Section 6.585](#) property specifies whether or not the user can select more than one button in a [Section 6.585](#) at the same time, by holding down the Ctrl key when clicking.

MultiSelect is a single number with the value 0 (only 1 button may be selected) or 1 (more than one button may be selected); the default is 0.

MultiSelect applies only if the [Section 6.577](#) of the [Section 6.585](#) is 'Buttons' or 'FlatButtons', and is ignored if [Section 6.577](#) is 'Tabs'.

Note that the [Section 6.571](#) property of the associated [Section 6.584](#) object reports whether or not the button is selected.

6.406 NameFromHandle Method 136

Applies To: [Section 6.489](#)

Description

This method is used to obtain the name of a particular object from the value of its Handle property.

The argument to NameFromHandle is a single item as follows:

[1]	Handle	The value of the Handle property from an existing object.

The result is a character vector containing the name of the object.

6.407 NetClient Object

Purpose: The NetClient object represents an instance of a Microsoft .NET class.

Parents NetType Root

Children Timer

Description

The NetClient object represents an instance of a .NET class.

Normally, you create a NetClient object using the New method. For example:

```

[USING ← 'System'
DT1←DateTime.New 2002 4 30
DT1.Type
NetClient

```

If, for any reason, you are unable to use the `New` method, you may create a `NetClient` object using `[WC`. In this case, the `ClassName` property specifies the *full* name of the .NET class, and the `ConstructorArgs` property specifies the arguments for the constructor function if required.

```

[USING ← 'System'
'DT2'[WC'NetClient' 'System.DateTime'(1949 4 30)
DT2.(Type ClassName ConstructorArgs)
NetClient System.DateTime 1949 4 30

```

6.408 NetType Object

Purpose: The `NetType` object is used to export a namespace as a Microsoft.NET class.

Parents Root

Children Bitmap NetClient TCPSocket Timer

Properties BaseClass

Methods

Events

Description

The `NetType` object allows you to export an APL namespace as a .NET class that can be accessed by any conforming .NET client application.

The `BaseClass` property specifies the name of the .NET class from which the specified `NetType` object inherits. The default is `System.Object`.

When you create a `NetType` object, the name of its parent namespace specifies the name of the corresponding Microsoft .NET Namespace to which the `NetType` class belongs. If the `NetType` is created as a child of root, the corresponding Microsoft .NET Namespace is unnamed.

6.409 NewLine Property

Applies To: [Section 6.126](#)

Description

The NewLine property specifies whether or not a [Section 6.126](#) occupies the same row as an existing [Section 6.126](#), or is displayed on a new line within its [Section 6.127](#) parent.

NewLine is a single number with the value 0 (same row) or 1 (new row); the default is 1.

The value of NewLine in the first [Section 6.126](#) in a [Section 6.127](#) is always 1, even if you specify it to be 0.

When the user drags a [Section 6.126](#) to another row, the value of its NewLine property, and that of any other [Section 6.126](#) affected by the move, will change.

You may move a [Section 6.126](#) to the previous or next row by changing its NewLine property (using `WS`) from 1 to 0, or from 0 to 1 respectively.

6.410 NewPage Method 102

Applies To: [Section 6.455](#)

Description

This method causes a [Section 6.455](#) to start a new page

The NewPage method is niladic.

If you attach a callback function to this event and have it return a value of 0, the page throw will not occur.

6.411 Note Property

Applies To: [Section 6.55](#)

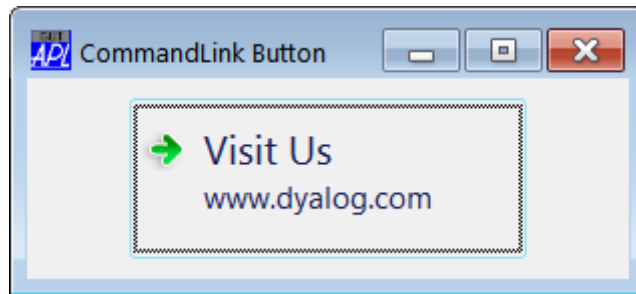
Description

The Note property applies only to a [Section 6.55](#) whose Style is 'CommandLink'.

It is a character vector (by default empty) that specifies text to be displayed below the Section 6.66.

EXAMPLE

```
'F'[]WC'Form' 'CommandLink Button'
'F.clb'[]WC'Button' 'Visit Us'('Style' 'CommandLink')
F.clb.Size←80 200
F.clb.Note←'www.dyalog.com'
```



6.412 OCXClass Object

Purpose: This object provides access to OLE (ActiveX) Controls.

Parents ActiveXControl CoolBand Form Grid OLEServer PropertyPage Root SubForm
TCPSocket ToolBar ToolControl

Properties Type ClassName Event Data Translate ClassID KeepOnClose TypeList HelpFile
ToolboxBitmap LicenseKey QueueEvents MethodList ChildList EventList PropList

Methods Detach GetPropertyInfo GetEventInfo GetMethodInfo GetTypeInfo
SetPropertyInfo SetMethodInfo ShowHelp ShowProperties Browse OLEAddEventSink
OLEDeleteEventSink OLEListEventSinks

Events

Description

This object loads an OLE Control into memory and defines a new class of object associated with it. The name of the new class is the name specified by the left argument of `[]WC`. You may create an instance of the newly defined class using the name you assigned to the OCXClass object as the Type property.

Note that you may not create an instance of OCXClass using `[]NEW`.

Once you have defined a new OCXClass, the properties, events and methods it supports may be obtained from its [Section 6.464](#), [Section 6.197](#) and [Section 6.387](#) properties. These are the properties, events and methods defined for the OLE control by its author.

The [Section 6.466](#) property determines how events reported by the ActiveX control are handled.

To find out how to use the OLE control, you must consult the appropriate documentation. However, a great deal of information about it can be obtained using the [Section 6.254](#), [Section 6.245](#), and [Section 6.251](#) methods.

6.413 OLEAddEventSink Method 540

Applies To: [Section 6.412](#), [Section 6.414](#)

Description

This method connects a named event sink to a COM object and adds the events defined by that event sink to the [Section 6.197](#) property of the associated namespace.

The argument to OLEAddEventSink is a single item as follows:

[1]	Event sink name	character vector

The result is a number that represents the handle of the event sink. This may be subsequently required.

6.414 OLEClient Object

Purpose: The OLEClient object provides access to an OLE Automation Server

Parents ActiveXControl CoolBand Form OLEServer Root TCPSocket

Children Form TCPSocket Timer

Properties Type ClassName Event Data Handle ClassID KeepOnClose TypeList HelpFile
LastError Locale AutoBrowse QueueEvents InstanceMode LateBind MethodList
ChildList EventList PropList

Methods Browse Detach GetEventInfo GetMethodInfo GetPropertyInfo GetTypeInfo
OLEAddEventSink OLEDeleteEventSink OLEListEventSinks OLEQueryInterface
SetMethodInfo SetPropertyInfo ShowHelp

Events

Description

The OLEClient object allows you to control OLE Servers, which may be written in a variety of different programming languages, including Dyalog APL itself.

The [Section 6.97](#) property specifies the name of the OLE object to which the new object named by the left argument of `⌈wc` is to be connected. A list of all the OLE Server objects installed on your system may be obtained from the OLEServers property of Root. [Section 6.97](#) may only be specified by `⌈wc`.

Alternatively, the OLE object may be identified by the [Section 6.96](#) property.

The AutoBrowse property and Browse method are no longer relevant and are ignored. They are retained only for backwards compatibility with previous versions of Dyalog APL.

Note that the PropList and MethodList properties of an OLEClient instance contain the names of the properties and methods exposed by the corresponding OLE Object in addition to the generic properties and methods of the OLEClient class.

If you call an OLE method with an invalid parameter, set a read-only property, or assign it an invalid value, the [Section 6.342](#) property of the OLEClient and [Section 6.489](#) objects will contain error information generated by OLE.

6.415 OLEControls Property

Applies To: [Section 6.489](#)

Description

The OLEControls property reports a list of the OLE Controls installed on your computer. This information is obtained from the Windows registry. Its value is a nested vector with one element per OLE Control. Each element is a vector of 2-element character vectors. The first is the name of the OLE Control; the second is its [Section 8.1](#).

6.416 OLEDeleteEventSink Method 541

Applies To: [Section 6.412](#), [Section 6.414](#)

Description

This method disconnects a named event sink from a COM object and removes the events defined by that event sink from the [Section 6.197](#) property of the associated namespace.

This method may be used to remove an event sink that was established automatically when the OLE object was created.

The argument to OLEDeleteEventSink is a single item as follows:

[1]	Event sink name	character vector

6.417 OLEListEventSinks Method 542

Applies To: [Section 6.412](#), [Section 6.414](#)

Description

This method returns the names of *event sinks* that are currently connected to a COM object.

The list contains the names of all the event sinks that were connected automatically when the object was created, together with any that you have added subsequently using [Section 6.413](#).

The OLEListEventSinks method is niladic.

The result is a vector of character vectors containing the names of the event sinks connected to the object.

6.418 OLEQueryInterface Method 543

Applies To: [Section 6.6](#), [Section 6.414](#)

Description

This method is used to obtain the methods and properties associated with a particular *interface* that is provided by a COM object. An interface is simply a pointer to a table of methods (not properties) that are exported by an object.

Note that methods and properties exported using the standard IDispatch interface are established automatically when the object is created. OLEQueryInterface is required only to support alternative or additional interfaces that the object may implement.

The argument to OLEQueryInterface is a single item as follows:

[1]	Interface name	character vector

The result is a namespace.

It is normal, although not strictly required, that the new namespace be a child of the one for which the method is run.

Note that if the object does not support a type library, the new namespace will be empty and you will have to establish functions corresponding to the methods exported by the interface using [Section 6.534](#).

6.419 OLERegister Method 530

Applies To: [Section 6.420](#)

Description

This method is used to register an [Section 6.420](#) object and may be used to install Dyalog OLE Servers as part of a run-time installation.

If the argument to the OLERegister method is a simple character vector, this is treated as text to be inserted into the command line argument for the interpreter. All other arguments are silently ignored.

Note

It is currently necessary to run the APL from which this method is called with Administrator privileges.

6.420 OLEServer Object

Purpose: The OLEServer object is used to establish a namespace as an OLE Server object that can be used by an OLE Automation client.

Parents ActiveXControl Form OLEServer Root

Children Bitmap BrowseBox Clipboard Cursor FileBox Font Form Icon ImageList Menu Metafile MsgBox OCXClass OLEClient OLEServer Printer PropertySheet TCPSocket Timer TipField

Properties Type ClassName Event Data Handle ExportedFns ExportedVars ClassID KeepOnClose TypeLibID TypeLibFile ServerVersion LastError RunMode ShowSession LateBind MethodList ChildList EventList PropList

Methods Detach OLERegister OLEUnregister SetFnInfo SetVarInfo SetEventInfo

Events Close Create

Description

The OLEServer object allows you to export an APL namespace so that its functions and variables become directly accessible to an OLE Automation client application such as Microsoft Visual Basic or Microsoft Excel.

An OLEServer may be saved as an *out-of-process* OLE server (in a workspace) or as an *in-process* OLE server (in a DLL). See *Interface Guide* for details.

When you create an OLEServer object, APL allocates various OLE attributes to it. For example, the CLSID, which uniquely identifies the object, is assigned at this stage. However, the object is not actually *registered* until you execute)SAVE.

Registration involves updating the Windows registry with information about the object itself, such as its name, the command required to obtain it and so forth. Registration also records information about all of the functions and variables that your object exposes. Registration is therefore a non-trivial operation and should be delayed until the point when you are ready to test your OLEServer.

You may create an empty OLEServer object and then define functions and variables within it. Alternatively, you may convert an existing namespace which is already populated with functions and variables. The latter method is recommended as it implies less registry activity during the development of the object.

The [Section 6.202](#) and [Section 6.203](#) properties specify the names of the functions and variables that will be exposed by the object to OLE clients.

The [Section 6.507](#) property is a character vector that specifies how the object serves multiple clients. It may be 'MultiUse' (the default), 'SingleUse', or 'RunningObject'.

The [Section 6.553](#) property is either 0 (the default) or 1 and specifies whether or not the APL Session window is displayed when the first instance of the OLEServer is created.

[Section 6.507](#) and [Section 6.553](#) apply only to *out-of-process* OLEServers.

6.421 OLEServers Property

Applies To: [Section 6.489](#)

Description

The OLEServers property is a read-only property that reports the names and CLSIDs of all the OLE Automation servers installed on your computer. This information comes from the Windows registry.

Its value is a nested vector with one element per OLE Server.

Each element is a vector of 2-element character vectors. The first is the name of the OLE Server; the second is its class identifier or CLSID which is a type of [Section 8.4](#).

6.422 OLEUnregister Method 531

Applies To: [Section 6.420](#)

Description

This method is used to unregister an [Section 6.420](#) object that has previously been saved by Dyalog APL.

The OLEUnregister method is niladic.

This method removes all traces of the object from the Windows registry and erases its Type Library file.

Note that the name of the object removed from the registry is the name of the [Section 6.420](#) object prefixed by the string "dyalog."

6.423 OnTop Property

Applies To

Description

This property may be used to cause a [Section 6.227](#) or [Section 6.578](#) to be displayed on top of all other windows, even those owned by other applications.

Normally, [Section 6.227s](#) are brought to the front when they receive the input focus. [Section 6.227s](#) that do not have the input focus may be partially obscured by the one that does. If OnTop is set to 1, the [Section 6.227](#) or [Section 6.578](#) remains at the front even if it doesn't have the input focus. Indeed, it may partially obscure the [Section 6.227](#) that does have the focus. The default value is 0 (normal).

More than one [Section 6.227](#) may have OnTop set to 1. If so, these [Section 6.227s](#) appear on top of all others, but may overlap one another. Other applications may also have windows with this property.

For a graphical object, the OnTop property controls how it is drawn in a [Section 6.263](#) relative to the grid lines and cell text. OnTop is applicable only if the graphic is the child of a [Section 6.263](#) and is otherwise ignored.

0	Graphical object is drawn behind grid lines and cell text
1	Graphical object is drawn on top of grid lines but behind cell text
2	Graphical object is drawn on top of grid lines and cell text

6.424 Orientation Property

Applies To: [Section 6.455](#)

Description

The Orientation property specifies the orientation of the paper on a [Section 6.455](#) object. It is a simple character vector which is either 'Portrait' or 'Landscape'. When you create a [Section 6.455](#) object, the default value of the Orientation property is determined by the current setting for the corresponding printer device.

The effect of changing Orientation using `□ws` is to spool the current page (effectively the same as sending a NewPage event) and then to change the orientation of the paper. Note that the values of the first 2 elements of the DevCaps property change accordingly. You may also set Orientation when you create the [Section 6.455](#) object with `□wc`. In neither case does the global setting for the printer device change.

6.425 OtherButton Property

Applies To: [Section 6.108](#)

Description

The OtherButton property is Boolean and specifies whether or not the user can select a colour from a [Section 6.108](#) object using the Windows colour selection dialog box.

If OtherButton is 1 (the default), the final row of the colour selection drop-down contains a button labelled "Other...". If the user clicks this button, the standard Windows colour selection dialog box is displayed, allowing the user to select any colour that the computer can render.

If OtherButton is 0, the button labelled "Other..." is not present and the user is restricted to the choice of colours provided by the [Section 6.149](#) property.

6.426 OverflowChar Property

Applies To: [Section 6.263](#)

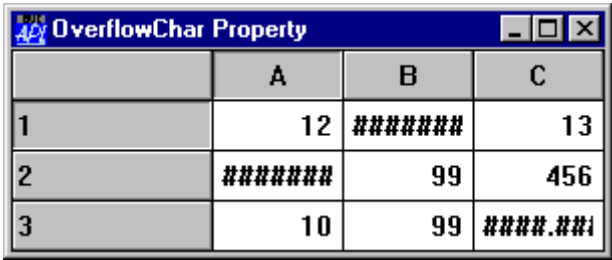
Description

The OverflowChar property specifies the character to be displayed in place of the digits when a numeric value cannot be displayed in its entirety in a Grid cell.

If the value of OverflowChar is an empty vector (the default) the data in a numeric cell is simply clipped if it is too wide to fit in the cell.

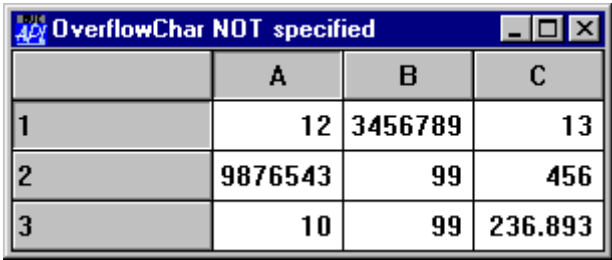
FOR EXAMPLE

```
'F'□WC'Form'('Coord' 'Pixel')('Size' 101 296)
'F'□WS'Caption' 'OverflowChar Property'
DATA←3 3p12 123456789 13 9876543 99 456 10 99 1236.893
'F.G'□WC'Grid'DATA(0 0)(101 296)
'F.G'□WS'CellWidths' 65
'F.G'□WS'OverflowChar' '#'
```

	A	B	C
1	12	#####	13
2	#####	99	456
3	10	99	####.###i

The same Grid without OverflowChar being defined appears as follows. Notice how the numbers have been truncated



	A	B	C
1	12	3456789	13
2	9876543	99	456
3	10	99	236.893

6.427 PageActivate Event 360

Applies To: [Section 6.462](#)

Description

If enabled, this event is reported when the user switches from one [Section 6.462](#) to another in a [Section 6.463](#) object. This event is reported by the new page *after* the page change has occurred and the page change may not be disabled by a callback function.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'PageActivate' or 360

You may select a particular page by calling PageActivate as a method, or by setting the [Section 6.428](#) or [Section 6.429](#) property of the [Section 6.463](#).

6.428 PageActive Property

Applies To: [Section 6.463](#)

Description

The PageActive property specifies the name of the current [Section 6.462](#) in a [Section 6.463](#). You may select a particular page by setting this property or by generating a [Section 6.427](#) event.

See also or [Section 6.429](#).

6.429 PageActiveObject Property

Applies To: [Section 6.463](#)

Description

The PageActiveObject property specifies a ref to the current [Section 6.462](#) in a [Section 6.463](#). You may select a particular page by setting this property or by generating a [Section 6.427](#) event.

See also or [Section 6.428](#).

6.430 PageApply Event 350

Applies To: [Section 6.462](#)

Description

If enabled, this event is reported when the user clicks the Apply button in a [Section 6.463](#). Note however, that the event is actually reported by each of its [Section 6.462](#) objects whose Changed property is currently 1, that is, the event is reported by each of the pages that the user has changed.

The default processing for this event is to set the Changed property of the [Section 6.462](#) to 0. If you disable the event or return a 0 from a callback function, the Changed property is not reset. Note that the Apply button in a [Section 6.463](#) is active if the value of the Changed property of *any* of the [Section 6.462](#) objects is 1.

The event message reported as the result of `□□□`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' PageAppLy ' or 350

6.431 PageBack Event 353

Applies To: [Section 6.462](#)

Description

If enabled, this event is reported when the user switches from one [Section 6.462](#) to another in a Wizard [Section 6.463](#) object by clicking its Back button. This event is reported by the old page *after* the page change has occurred and the page change may not be disabled by a callback function.

The event message reported as the result of □□□, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' PageBack ' or 353

6.432 PageCancel Event 351

Applies To: [Section 6.462](#)

Description

If enabled, this event is reported when the user presses the Cancel button in a [Section 6.463](#) object and is reported by the current [Section 6.462](#). This event is reported for information only and may not be disabled by a callback function. However, the operation will also generate a Close event reported by the [Section 6.463](#) itself that may be disabled by a callback.

The event message reported as the result of □□□, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' PageCancel ' or 351

6.433 PageChanged Event 356

Applies To: [Section 6.462](#)

Description

If enabled, this event is reported when the [Section 6.86](#) property of a [Section 6.462](#) is altered by user action. It is *not* reported if you reset the [Section 6.86](#) property using `WS`.

The [Section 6.86](#) property is reset by two separate user actions. It is set to 1 when the user alters any of the controls on the [Section 6.462](#). It is reset to 0 when the user clicks the Apply button, although this action may be disabled by a callback function on the [Section 6.430](#) event.

The PageChanged event is reported for information only and may not itself be disabled or affected by a callback function.

The event message reported as the result of `WDQ`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'PageChanged' or 356
[3]	Changed value	New value for the Changed property (0 or 1).

6.434 PageDeactivate Event 361

Applies To: [Section 6.462](#)

Description

If enabled, this event is reported when the user switches from one [Section 6.462](#) to another in a [Section 6.463](#) object. This event is reported by the old page *after* the page change has occurred and the page change may not be disabled by a callback function.

The event message reported as the result of `WDQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'PageDeactivate' or 361

6.435 PageFinish Event 355

Applies To: [Section 6.462](#)

Description

If enabled, this event is reported when the user clicks the Finish button in a Wizard [Section 6.463](#). This event is reported by current (last) [Section 6.462](#). The event is reported for information only and cannot be affected by a callback function.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'PageFinish' or 355

6.436 PageHelp Event 352

Applies To: [Section 6.462](#)

Description

If enabled, this event is reported when the user clicks the Help button in a Wizard [Section 6.463](#). This event is reported by current [Section 6.462](#). The event is reported for information only and cannot be affected by a callback function.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'PageHelp' or 352

6.437 PageNext Event 354

Applies To: [Section 6.462](#)

Description

If enabled, this event is reported when the user switches from one [Section 6.462](#) to another in a Wizard [Section 6.463](#) object by clicking its Next button. This event is

reported by the old page *after* the page change has occurred and the page change may not be disabled by a callback function.

The event message reported as the result of `□□□`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' PageNext ' or 354

6.438 PageSize Property

Applies To: [Section 6.227](#), [Section 6.508](#), [Section 6.578](#)

Description

For a [Section 6.227](#) and [Section 6.578](#), the PageSize property is a 2-element integer vector which specifies the size of the thumb in the vertical and horizontal scrollbars respectively.

For a [Section 6.508](#) object it is a single integer.

If PageSize is 0 (the default) it specifies the default thumb. Otherwise, PageSize is expressed in proportion to the corresponding value of [Section 6.469](#). For example, if [Section 6.469](#) is 1000, setting PageSize to 100 will obtain a thumb which is approximately 10% of the height or length of the scrollbar.

6.439 PageWidth Property

Applies To: [Section 6.488](#)

Description

The PageWidth property specifies the width of the page in a [Section 6.488](#) object and is the dimension that is used to apply text wrapping and paragraph formatting to the text in the object. PageWidth is a single integer value specified in [Section 8.9](#).

The default value of PageWidth is 0 which disables text wrapping. You may find it convenient to set PageWidth to the width of the [Section 6.488](#) window or to a value that is appropriate for your printer.

6.440 PaperSize Property

Applies To: [Section 6.455](#)

Description

The PaperSize property specifies the size of paper to be used for printing

PaperSize may be a character vector containing the name of the paper size (for example, 'Legal 8 1/2 x 14 in' or 'A4 210 x 297 mm') or a 2-element integer vector that specifies the desired height and width of the paper in tenths of a millimetre (for example, 3556 2159 or 2970 2099).

The default value of PaperSize is the name of the paper size associated with the current printer settings.

You can obtain a list of supported paper sizes from the [Section 6.441](#) property.

6.441 PaperSizes Property

Applies To: [Section 6.455](#)

Description

The PaperSizes property is a read-only property that provides the names and dimensions of the various different paper sizes supported by the printer associated with the Printer object.

PaperSizes is a nested vector of 2-element vectors which contain the name, and height and width of each paper size respectively. Dimensions are reported in tenths of a millimetre.

You may set or query the current paper size using the [Section 6.440](#) property.

6.442 PaperSource Property

Applies To: [Section 6.455](#)

Description

The PaperSource property is a character vector that specifies the name of the paper bin to be used as the paper source for printing.

An empty character vector (the default) means the default bin, Otherwise, PaperSource must be a member of the [Section 6.443](#) property.

6.443 PaperSources Property

Applies To: [Section 6.455](#)

Description

The PaperSources property is a read-only property that provides the names of the paper bins installed on the printer associated with the Printer object. It is a vector of character vectors.

You may select which of the bins is to be used by specifying the [Section 6.442](#) property.

6.444 ParaFormat Property

Applies To: [Section 6.488](#)

Description

The ParaFormat property describes the current paragraph format or the paragraph format of the currently selected text in a [Section 6.488](#) object. It is a 6-element nested array structured as follows:

[1]	A character vector that specifies the text alignment. This may be 'Left' (the default), 'Right' or 'Centre'
[2]	The size of the indentation of the first line in the paragraph measured from the left margin in <i>Twips</i> .
[3]	The size of the horizontal offset of the start of the second and subsequent lines. This is measured in <i>Twips</i> relative to the first line indentation specified in element [2].
[4]	The size of the right indentation measured in <i>Twips</i> from the right margin.
[5]	An integer value specifying the bullet/numbering option. 0 mean no numbering, 1 means bullets.
[6]	An integer vector specifying the size of any tab stops measured in <i>Twips</i> from the left margin and specified in ascending order.

If there is no text selected, ParaFormat specifies the *current* paragraph formatting format, that is, that which will be used to format the current (and subsequent) lines of characters that the user enters. If there is text selected ParaFormat specifies the paragraph formatting of the selected block of text. If the format is not strictly homogeneous, ¶WG will report the format of the first paragraph in the selected block

(¶WS 'ParaFormat' ...) will set the format of the currently selected block of text. To set the format of an arbitrary block of text you must select it first using (¶WS 'SelText' ...).

6.445 Password Property

Applies To: [Section 6.56](#), [Section 6.185](#), [Section 6.563](#)

Description

This property specifies the character that is echoed when a user enters data into a single-line [Section 6.185](#) object ([Section 6.577](#) 'Single'). It does not apply to a multi-line object ([Section 6.577](#) 'Multi'). If Password is empty (the default) the character echoed is the same as the character the user entered. If Password is set to (say) the asterisk character (*), the object will display asterisks as the user types into it.

6.446 PathWordBreak Property

Applies To: [Section 6.122](#)

Description

If set, the edit control portion of the ComboBox will use the forward slash (/), back slash (\), and period (.) characters as word delimiters. This makes keyboard shortcuts for word-by-word cursor movement (Ctrl + arrow keys) effective in path names and URLs.

6.447 Picture Property

Applies To

Description

The Picture property specifies a bitmap, icon, or other image for an object.

For [Section 6.55](#), [Section 6.227](#), [Section 6.279](#), [Section 6.378](#), [Section 6.573](#), [Section 6.574](#), [Section 6.575](#), [Section 6.578](#), [Section 6.558](#), [Section 6.582](#) or [Section 6.621](#), this

property specifies the name of, or ref to, a [Section 6.47](#), [Section 6.309](#), or [Section 6.385](#) which is drawn as a *background* on the object. Other controls and graphical objects are drawn on top of this background.

When it refers to a [Section 6.385](#), the Picture property specifies the name of, or ref to, the [Section 6.385](#) to be drawn in the object. When it refers to a [Section 6.47](#) or [Section 6.309](#), the value of the Picture property is a 2-element vector whose elements specify the name of, or ref to, the [Section 6.47](#), or [Section 6.309](#), and the manner in which it is displayed. This is specified as an integer as follows:

0	The Bitmap or Icon is drawn in the top left corner of the object.
1	The Bitmap or Icon is tiled (replicated) to fill the object.
2	The Bitmap is scaled (up or down) to fit exactly in the object. This setting does not apply to an Icon whose size is fixed.
3	The Bitmap or Icon is drawn in the centre of the object. This is the default. Note that the centre of the Bitmap is positioned over the centre of the object, so that you see the middle portion of a Bitmap that is larger than the object in which it is displayed.

For example, the following statements produce a [Section 6.227](#) filled with the CARS bitmap.

```
'CARS' □WC 'Bitmap' 'C:\WINDOWS\CARS'
'f1' □WC 'Form' ('Picture' 'CARS' 1)
```

An easy way to provide a customised pushbutton is to create a [Section 6.55](#) whose Picture property specifies the name of, or ref to, a [Section 6.47](#) or [Section 6.309](#), using drawmode 3 (the default). This causes the corresponding bitmap or icon to be drawn in the centre of the [Section 6.55](#). So long as the [Section 6.55](#) is larger than the bitmap or icon, its borders (which give it its 3-dimensional appearance and "pushbutton" behaviour) will be unaffected.

Note that if Picture is set on a [Section 6.55](#) whose [Section 6.577](#) is 'Radio' or 'Check', the Button assumes pushbutton appearance, although its radio/check behaviour is preserved.

For an [Section 6.313](#) object, the Picture property specifies the name of, or ref to, a [Section 6.47](#), [Section 6.309](#) or [Section 6.385](#) object to be drawn, or a vector of names or refs. The [Section 6.313](#) is a graphical object and is drawn *on top of* the background. It does not support the drawmode options provided by the objects in which Picture specifies the background.

For the [Section 6.99](#) object, Picture is a "set-only" property that allows you to place a specified [Section 6.47](#) object into the Windows clipboard. To place a [Section 6.385](#) object into the clipboard, use its [Section 6.385](#) property.

6.448 PName Property

Applies To: [Section 6.222](#), [Section 6.455](#)

Description

This property is a character vector that specifies the face name for a [Section 6.222](#) object, or the printing device associated with a [Section 6.455](#). It is case-independent.

For a [Section 6.455](#), PName contains the description of the printer followed by a comma (,) and then the device to which it is attached.

EXAMPLE

```
'PR1' □WC 'Printer'
'PR1' □WC 'PName'
HP Universal Printing PS, hp4200
```

6.449 Points Property

Applies To

Description

This property specifies the co-ordinates for a graphics object. It may define a single set of co-ordinates, or be a nested scalar or a nested vector each element of which contains a set of co-ordinates.

Each set of co-ordinates may be:

- a 2-column numeric matrix containing y-values in column 1 and x-values in column 2. **or**
- a 2-element numeric vector whose first element specifies y-values and whose second element specifies x-values. The two elements must be of equal length unless one or both is a scalar in which case scalar extension applies.

For further details, see the specifications for the relevant objects.

6.450 Poly Object

Purpose: A graphical object used to draw lines, polygons, and filled areas.

Parents ActiveXControl Animation Bitmap Button ButtonEdit Combo ComboEx Edit Form Grid Group Label List ListView MDIClient Metafile Printer ProgressBar PropertyPage PropertySheet RichEdit Scroll Spinner Static StatusBar SubForm TabBar TipField ToolBar TrackBar TreeView UpDown

Children Timer

Properties Type Points FCol BCol LStyle LWidth FStyle FillCol Coord Visible Event Dragable OnTop CursorObj AutoConf Data Accelerator KeepOnClose DrawMode MethodList ChildList EventList PropList

Methods Detach

Events Close Create DragDrop MouseDown MouseUp MouseMove MouseDbClick Help Select

Description

The [Section 6.449](#) property specifies one or more sets of co-ordinates through which one or more lines are drawn. The resulting polygon(s) may also be filled.

[Section 6.360](#) and [Section 6.361](#) define the style and width of the lines. [Section 6.205](#) and [Section 6.45](#) determine the colour of the lines.

[Section 6.231](#) specifies whether or not the polygon(s) are filled, and if so, how. For a solid fill ([Section 6.231](#) 0), [Section 6.214](#) defines the fill colour used. For a pattern fill ([Section 6.231](#) 1-6) [Section 6.214](#) defines the colour of the hatch lines and [Section 6.45](#) the colour of the areas between them.

Note that if you specify filling, you do not have to define a **closed** polygon. The first and last points will automatically be joined for you if necessary.

The value of [Section 6.173](#) determines whether or not the object can be dragged. The value of [Section 6.35](#) determines whether or not the Poly object is resized when its parent is resized.

The structure of the property values is best considered separately for single and multiple polylines or polygons.

Single Polyline or Polygon

For a single polyline or polygon, [Section 6.449](#) is either a 2-column matrix of (y,x) co-ordinates, or a 2-element vector of y and x co-ordinates respectively.

[Section 6.360](#) and [Section 6.361](#) are both simple scalar numbers.

[Section 6.231](#) is either a single number specifying a standard fill pattern, or the name of a [Section 6.47](#) object which is to be used as a "brush" to fill the polygon.

[Section 6.205](#), [Section 6.45](#) and [Section 6.214](#) are each either single numbers representing standard colours, or 3-element vectors which specify colours explicitly in terms of their RGB values.

EXAMPLES

First make a [Section 6.227](#) :

```
'F' WC 'Form'
```

Draw a single line from (y=20, x=10) to (y=30, x=50)

```
'F.L1' WC 'Poly' ((20 30)(10 50))
```

or

```
L ← 2 2p20 10 30 50
'F.L1' WC 'Poly' L
```

Draw a horizontal line from (y=20, x=10) to (y=20, x=50). Note scalar extension of y-coordinate.

```
'F.L1' WC 'Poly' (20(10 50))
```

Draw an empty box in green :

```
Y ← 10 10 50 50 10
X ← 10 50 50 10 10
'F.L1' WC 'Poly' (Y X) (0 255 0)
```

Ditto, using a green/blue dashed line ([Section 6.360](#) 1) :

```
'F.L1' WC 'Poly' (Y X) (0 255 0)(0 0 255) 1
```

Draw a red filled rectangle with a black border 5 pixels wide :

```
'F.L1' WC 'Poly' (Y X) (0 0 0) ('LWidth' 5)
('FStyle' 0)('FillCol' 255 0 0)
```

Multiple Polylines/Polygons

To draw a set of polylines or polygons with a single name, [Section 6.449](#) is a nested vector whose items are themselves 2-column matrices or 2-element nested vectors.

[Section 6.360](#) and [Section 6.361](#) may each be simple scalar values (applying to all the polylines) or simple vectors whose elements refer to each of the corresponding polylines in turn.

[Section 6.231](#) may be a simple scalar numeric or a simple character vector ([Section 6.47](#) name) applying to all polylines, or a vector whose elements refer to each of the corresponding polylines in turn.

Similarly, [Section 6.205](#), [Section 6.45](#) and [Section 6.214](#) may each be single numbers or a single (enclosed) 3-element vector applying to all the polylines. Alternatively, these properties may contain vectors whose elements refer to each of the polylines in turn. If so, their elements may be single numbers or nested RGB triplets, or a combination of the two.

EXAMPLES

First make a [Section 6.227](#) :

```
'F' WC 'Form'
```

Draw two concentric triangles :

```
BY ← 10 10 50 10
BX ← 15 65 40 15
RY ← 15 15 40 15
RX ← 25 55 40 25
'F.L1' WC 'Poly' ((BY BX)(RY RX))
```

Or, using matrices :

```
BT ← BY,[1.5]BX
RT ← RY,[1.5]RX
'F.L1' WC P←'Poly' (BT RT)
```

Ditto, but draw the first blue, the second red :

```
'F.L1' WC P, c((0 0 255)(255 0 0))
```

Ditto, but make the lines 3 pixels wide :

```
'F.L1' □WC P, ((0 0 255)(255 0 0))('LWidth' 3)
```

Ditto, but make the line widths 3 and 6 pixels respectively :

```
'F.L1' □WC P, ((0 0 255)(255 0 0))('LWidth' 3 6)
```

Draw the first hollow, but fill the second in green :

```
'F.L1' □WC P, ('FStyle' -1 0)('FillCol' (=0 255 0))
```

6.451 Popup Property

Applies To: [Section 6.581](#), [Section 6.623](#)

Description

The Popup property specifies the name of, or ref to, a (popup) [Section 6.382](#) object that is associated with a [Section 6.581](#) or [Section 6.623](#).

Note that Popup is ignored unless [Section 6.577](#) is set to 'DropDown'.

6.452 Posn Property

Applies To

Description

With the exception of [Section 6.382](#), [Section 6.384](#) and [Section 6.522](#) objects, Posn is a 2-element numeric vector specifying the y-position and x-position respectively of the top-left corner of the object relative to its parent. For a [Section 6.227](#), Posn specifies its position on the screen. The units are defined by the [Section 6.128](#) property.

When specifying Posn for *Dyalog APL Language: Wc*, you can allow the y-position or x-position to assume a default value by giving the corresponding element a value of $\emptyset$.

Using *Dyalog APL Language: Ws*, if you want to set the y-position, but not the x-position, or vice-versa, you should specify $\emptyset$ for the item you don't want to change.

For [Section 6.382](#), [Section 6.384](#) and [Section 6.522](#) objects, Posn is a single integer that specifies the position at which the object is to be **inserted** in its parent. For example, to add a new [Section 6.384](#) between the third and fourth items in an existing [Section](#)

[6.382](#), you would specify its Posn as 4. For these objects, the value of Posn returned by *Dyalog APL Language: Wg* is the current index of the object within its parent.

6.453 PreCreate Event 534

Applies To: [Section 6.7](#)

Description

If enabled, this event is reported when an instance of an [Section 6.7](#) is created. The PreCreate event is generated at the point the *instance* is made.

An ActiveXControl also generates a [Section 6.131](#) event, which occurs *after* the PreCreate event at the point when the host application requires the instance to appear visually.

Note that at the time that PreCreate is generated, the [Section 6.7](#) does not have a window.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to -1 or by returning 0 from a callback function.

The event message reported as the result of □DQ, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'PreCreate' or 534

6.454 Print Method 100

Applies To: [Section 6.455](#)

Description

This method causes any spooled output to be printed.

The Print method is niladic.

If you attach a callback function to this event and have it return a value of 0, the printout will not be spooled.

6.455 Printer Object

Purpose: To provide printer output.

Parents ActiveXControl CoolBand Form OLEServer PropertyPage PropertySheet Root TCPSocket

Children Bitmap Circle Ellipse Font Icon Image Marker Metafile Poly Rect Text Timer

Properties Type PName DevCaps Coord Event FontObj FontList YRange XRange Data TextSize EdgeStyle Handle Orientation Copies PrintRange Collate PaperSize PaperSizes PaperSource PaperSources ColorMode Resolution Resolutions Duplex Translate Accelerator KeepOnClose MethodList ChildList EventList PropList

Methods Detach Print Setup NewPage Abort RTFPrintSetup GetTextSize

Events Close Create Select

Description

The [Section 6.448](#) property is a character vector which specifies the name of an installed printer and the device to which it is attached. The name and device are separated by a comma (,). All valid values of [Section 6.448](#) can be obtained from the [Section 6.456](#) property of the [Section 6.489](#) object.

If not specified, the default value of [Section 6.448](#) is (>'. ' □WG 'PrintList').

The [Section 6.158](#) property reports the size of the printable area of the page in pixels (dots) and in millimetres. It also reports the number of colours available. This is 2 on a monochrome printer (black and white), although grey scales may be available.

The [Section 6.224](#) property provides a list of fonts that are applicable and includes TrueType and printer fonts. This list is typically different from that obtained from the [Section 6.224](#) property on the [Section 6.489](#) object which lists those fonts that apply to the screen.

The Orientation property specifies the orientation of the page and may be either 'Portrait' or 'Landscape'.

The graphics objects listed above may be printed in much the same way as they may be displayed on a [Section 6.227](#) or [Section 6.573](#). The differences are :

Once an object has been created, it will be printed, even if its name is subsequently expunged.

An object does not **replace** an existing one which has the same name.

The act of changing one or more properties of a named object causes the object to be printed a second time. For example, changing the Section 6.452 of an object will print it again at a different place.

In general it is recommended that you use unnamed objects for printing.

The Printer object five methods :

Name	Event	Description
<u>Print</u>	100	Sends output to print spooler
<u>Setup</u>	101	Displays Printer Set-up dialog box
<u>NewPage</u>	102	Throws a new page
<u>Abort</u>	103	Aborts the print job
<u>RTFPrintSetup</u>	460	Displays Printer Set-up dialog box

EXAMPLES

Start a print job on the default printer

```
'PR1' □WC 'Printer'
```

Write a centred heading at the top of the page using a proportional font

```
'PR1.' □WC 'Text' 'Report Title' (0 50)('HAlign' 1) ('FontObj'
'Roman' 64)
```

Draw a line across the page, 2 pixels wide

```
'PR1.' □WC 'Poly' (2(0 100)) ('LWidth' 2)
```

Print a character matrix. Note that a fixed width font is used.

```
REPORT ← 'I6' □FMT ?20 6p1000
'PR1.' □WC 'Text' REPORT (10 0)('FontObj' 'DyalogAPL')
```

Throw a new page

```
PR1.NewPage
```

Spool output

□EX 'PR1'

6.456 PrintList Property

Applies To: [Section 6.489](#)

Description

This property provides a list of the printers that are installed on your computer system, that is, those listed when you select "printers" from the MS-Windows Control Panel. It is a "read-only" property of the [Section 6.489](#) object ' . '.

PrintList is a vector of character vectors. Each item in PrintList contains the name of an installed printer followed by a comma (,) and then the name of the device to which it is attached. The first item in PrintList is the default system printer.

EXAMPLE

```
p'. '□WG'PrintList'
6
]display >'.'□WG'PrintList'
→
|KODAK ESP-3 AiO,USB001|
↑'. '□WG'PrintList'
KODAK ESP-3 AiO,USB001
Send To OneNote 2007,Send To Microsoft OneNote Port:
Microsoft XPS Document Writer,XPSPort:
Microsoft Office Document Image Writer,Microsoft Document Imaging
Writer Port:
Fax,SHRFAX:
Auto Canon MP600 Printer on DIMENSION5150,\
\DIMENSION5150\Canon
```

6.457 PrintRange Property

Applies To: [Section 6.455](#)

Description

The PrintRange property specifies the range of pages to be printed.

PrintRange may be an empty character vector (the default), or 'All', either of which will cause all pages to be printed.

Alternatively, PrintRange may be a 3 or 4-element nested array whose items are:

[1]	'Pages'
[2]	Start page (integer)
[3]	End page (integer)
[4]	Maximum number of pages (integer)

In this case, printing starts at the page specified to be the Start page, and ends at the page specified by End page or after the Maximum number of pages has been reached, whichever is sooner.

6.458 PrintToPDF Method 845

Applies To: [Section 6.307](#)

Description

This method writes the content displayed in an [Section 6.307](#) object to a specified file in Portable Document Format (pdf).

The argument to PrintToPDF is a simple character scalar or vector containing a file name. Note that the method does not add any extension to the file name that is supplied.

The method returns a Boolean result which indicated whether or not the operation succeeded. If the file name contains a directory path, the path must already exist. The user must have permission to write the file.

6.459 ProgressBar Object

Purpose: The ProgressBar object is used to indicate the progress of a lengthy operation.

Parents ActiveXControl CoolBand Form Group PropertyPage StatusBar SubForm
ToolBar ToolControl

Children Bitmap Circle Cursor Ellipse Font Icon Marker Poly Rect Text Timer

Properties Type Posn Size Style Coord Active Visible Event Thumb Step Wrap Limits
 Sizeable Dragable BCol CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj
 Tip TipObj Translate Accelerator AcceptFiles KeepOnClose ProgressStyle Redraw
 TabIndex Interval MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ProgressStep

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose
 Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick
 MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate
 GestureTwoFingerTap GesturePressAndTap Select

Description

The ProgressBar object is a window that an application can use to indicate the progress of a lengthy operation. The appearance of the bar in the ProgressBar is determined by the [Section 6.461](#) property.

If ProgressStyle is Normal or Smooth, the size of the bar, intended to indicate the amount of progress, is determined using the [Section 6.608](#) property in relation to its [Section 6.345](#) property, and/or using the [Section 6.460](#) method. This can be updated as appropriate in the application logic or by using a [Section 6.613](#).

The range of a ProgressBar is specified by the [Section 6.345](#) property. This is a 2-element integer vector defining its minimum and maximum values. The position of the filled rectangle is specified by the Thumb property. You can update the ProgressBar by using `□WS` to set the value of the [Section 6.608](#) directly, or by using the [Section 6.460](#) method. The latter causes the [Section 6.608](#) to be updated by the value of the [Section 6.576](#) property.

If you attempt to set the [Section 6.608](#) to a value greater than its maximum value (using either method) the behaviour depends upon the value of the [Section 6.660](#) property which is Boolean and has a default value of 1. If [Section 6.660](#) is 1, the value obtained when you set the Thumb property is given by the expression:

$$\text{LIMITS}[1] + (1 + \text{LIMITS}[2] - \text{LIMITS}[1]) \mid \text{THUMB} - \text{LIMITS}[1]$$

where THUMB is the value to which you set the Thumb property and LIMITS is the value of the [Section 6.345](#) property. This causes the highlighted rectangle to begin filling again from the left.

If ProgressStyle is Marquee, the size of the bar is fixed and its position changes with time according to the value of the [Section 6.327](#) property. The values of [Section 6.608](#), [Section 6.345](#), [Section 6.660](#) and [Section 6.576](#) are irrelevant.

6.460 ProgressStep Method 250

Applies To: [Section 6.459](#)

Description

This method is used to increment the thumb in a [Section 6.459](#) object.

The ProgressStep method is niladic.

The ProgressStep method causes the [Section 6.459](#) to attempt to increment its thumb by the value of its [Section 6.576](#) property, taking into account the settings of its [Section 6.345](#) and [Section 6.660](#) properties.

If the values of the [Section 6.608](#), [Section 6.576](#) and [Section 6.345](#) properties are THUMB, STEP and LIMITS respectively, the new value of Thumb (and the corresponding position of the highlighted bar) is:

if Wrap is 0:

```
LIMITS[2][THUMB+STEP
```

if Wrap is 1:

```
LIMITS[1]+(1+LIMITS[2]-LIMITS[1])|THUMB+STEP-LIMITS[1]
```

6.461 ProgressStyle Property

Applies To: [Section 6.459](#)

Description

The ProgressStyle property specifies the appearance of a [Section 6.459](#) control.

ProgressStyle is a character vector that may be 'Normal', 'Smooth' or 'Marquee'. Its value is effective only when the object is created with `WC`. Changing ProgressStyle with `WS` has no effect on the appearance or behaviour of the [Section 6.459](#).

If ProgressStyle is 'Normal', the highlight in the centre of the [Section 6.459](#) is displayed as a broken bar. This is the default.

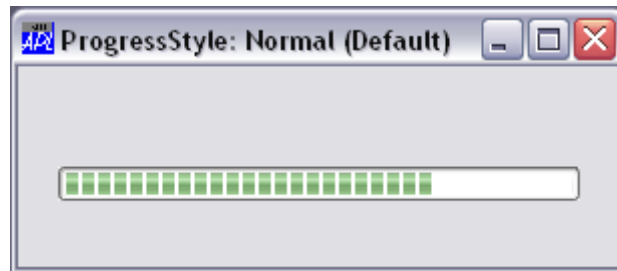
If ProgressStyle is 'Smooth', the highlight in the centre of the [Section 6.459](#) is displayed as a solid block of colour. This style only applies if *Windows Classic Theme* is in use. If not, it will be as if 'Normal' were specified.

If `ProgressStyle` is 'Marquee', the highlight in the centre of the [Section 6.459](#) is displayed as a broken bar that moves continuously from left to right. The speed is controlled by the [Section 6.327](#) Property which determines the frequency in milliseconds with which the highlight is redrawn, each time further along the [Section 6.459](#). The special value of `-1` causes the animation to stop.

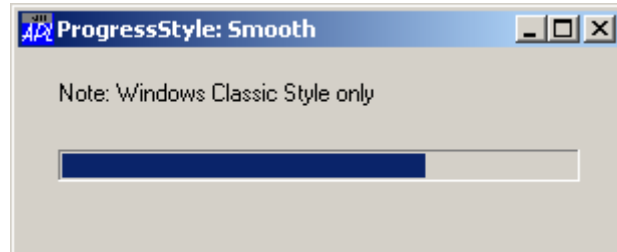
****Note that this feature only applies if Native Look and Feel**

is enabled.** If not, 'Marquee' will produce the same behaviour as 'Normal'.

The pictures below illustrate the appearance of the different values of `ProgressStyle`.



ProgressStyle Normal (the default)



ProgressStyle Smooth (Windows Classic Theme only)



ProgressStyle Marquee (requires Native Look and Feel)

6.462 PropertyPage Object

Purpose: The PropertyPage object represents a single page in a PropertySheet.

Parents PropertySheet

Children Animation Bitmap BrowseBox Button ButtonEdit Calendar Circle Clipboard ColorButton Combo ComboEx Cursor DateTimePicker Edit Ellipse FileBox Font Grid Group HTMLRenderer Icon Image ImageList Label List ListView Locator Marker Metafile MsgBox OCXClass Poly Printer ProgressBar Rect RichEdit Scroll SM Spinner Splitter Static SubForm TCPSocket Text Timer TipField TrackBar TreeView UpDown

Properties Type Caption Posn Size Coord Active Event HasHelp FontObj Data EdgeStyle Hint HintObj Tip TipObj Changed Translate AcceptFiles KeepOnClose MethodList ChildList EventList PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDblClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap FontOK FontCancel PageApply PageCancel PageHelp PageBack PageNext PageFinish PageChanged PageActivate PageDeactivate

Description

The PropertyPage object represents a single page within a [Section 6.463](#).

The Posn and Size properties are read-only properties determined by the parent [Section 6.463](#) and may not be changed using `WC` or `WS`.

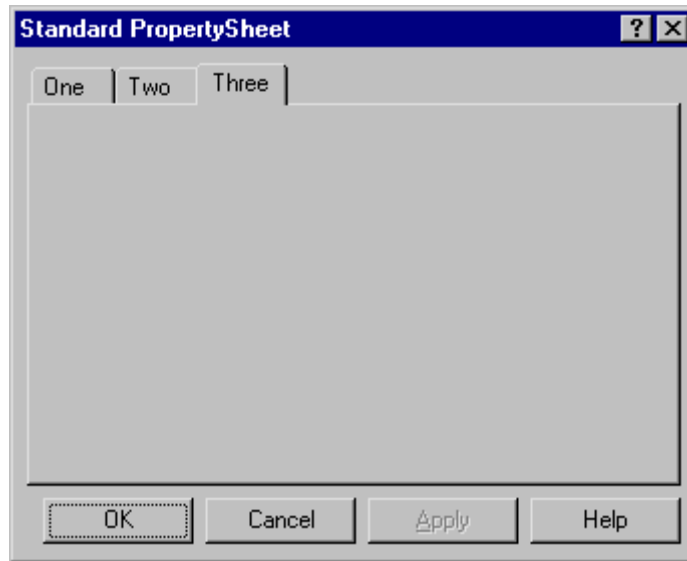
The [Section 6.287](#) property is either 1 (the default) or 0. If the parent [Section 6.463](#) has a "Help" button (determined by its own [Section 6.287](#) property) this property determines whether or not the Help button is active when the PropertyPage is the current page. If the [Section 6.287](#) property of a PropertyPage is 0, the Help button on the parent [Section 6.463](#) will be temporarily disabled when that PropertyPage is displayed.

The PropertyPage object generates a [Section 6.427](#) event when it becomes the current page and a [Section 6.434](#) event when another page is selected. These events may not be disabled by a callback function.

If the user presses the Cancel button, the current PropertyPage generates a [Section 6.432](#) event. This is followed by a Close event which is reported by the parent PropertySheet.

Other properties and behaviour depend upon the Style of the parent PropertySheet which may be 'Standard' or 'Wizard'

Standard Behaviour



In a *Standard* PropertySheet, the Caption property of each PropertyPage specifies the text that is written in its tab.

PropertyPage objects owned by a Standard [Section 6.463](#) generate [Section 6.432](#), [Section 6.430](#) and [Section 6.436](#) events. These events are all caused by the user pressing the corresponding button in the parent [Section 6.463](#).

Conventionally, the Apply button is initially inactive. When the user changes an item on any of the PropertyPages, the Apply button immediately becomes active. When the user clicks the Apply button, the application responds (normally by changing the appropriate properties) and then the Apply button becomes inactive once again. This process is controlled as follows.

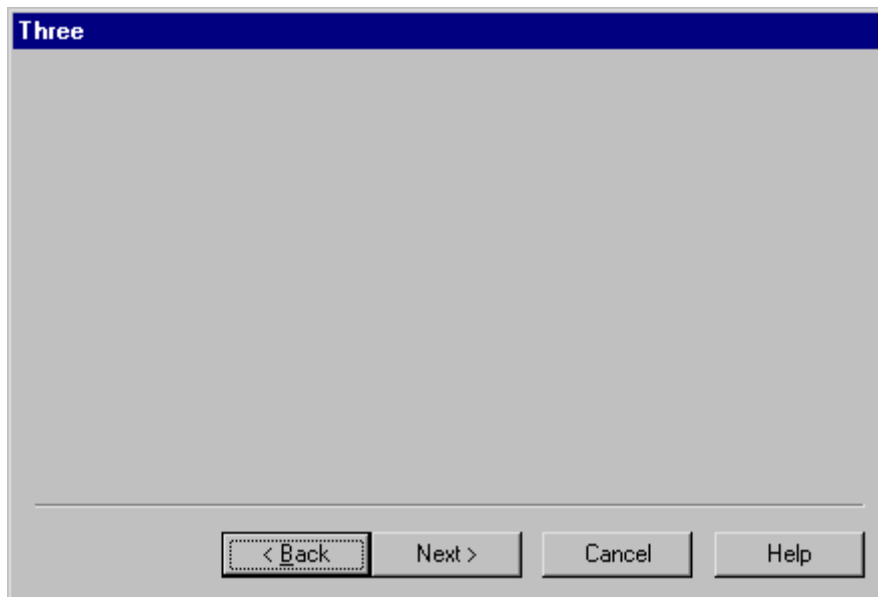
The Changed property is a Boolean value that determines whether or not a PropertyPage is marked as having been in any way altered. The Apply button is active if the value of the Changed property for *any* of the PropertyPages is 1, and is inactive otherwise

Initially, the value of the Changed property for all of the PropertyPages is 0 and the Apply button is therefore inactive. If the user alters a control on a PropertyPage, by, for example typing into an Edit object or changing the State of a Radio Button, the PropertyPage immediately generates a [Section 6.433](#) event with the parameter 1. The default processing for this event is to set the Changed property of the PropertySheet (to 1). This in turn activates the "Apply" button. If you return 0 from a callback on the [Section 6.433](#) event, the Changed property remains 0 and the Apply button remains inactive.

When the user clicks the Apply button, each of the PropertyPages whose Changed flag is currently set to 1 generates a [Section 6.430](#) event. The default processing for this event is to generate a [Section 6.433](#) event with the parameter 0. This in turn resets the Changed property of the PropertyPage to 0. Once all of the Changed flags have been reset, the Apply button becomes inactive. If you return 0 from a callback on any of the [Section 6.433](#) events, the Changed property for the corresponding PropertyPage remains 1 and the Apply button remains active.

You may control the value of the Changed property using `PPWS` or by calling [Section 6.433](#) as a method. In all cases, the Apply button is active if the value of Changed on any PropertyPage is 1, and inactive otherwise.

Wizard Behaviour



If the PropertyPage is owned by a Wizard [Section 6.463](#), its Caption property specifies the text that appears in the title bar of the [Section 6.463](#) window when the

PropertyPage is the current page. Note that a Wizard [Section 6.463](#) ignores its own Caption property.

There are effectively 3 page changing buttons on a Wizard [Section 6.463](#), named Back, Next and Finish. The Next and Finish buttons actually occupy the same position and are mutually exclusive. The captions on the buttons are language-dependent.

Conventionally, the buttons change according to which of the PropertyPages is currently displayed. If the first one is displayed, the Next button is active but the Back button is inactive. When a middle page is displayed, both the Next and Back buttons are active. When the last page is displayed, the caption on the Next button changes to Finish. However, in some applications, the Back button may be disabled to prevent the user returning to a previous page.

When the user clicks the Back or Next button, the PropertyPage generates a [Section 6.431](#) or [Section 6.437](#) event followed by a [Section 6.434](#) event. The new PropertyPage then generates a [Section 6.427](#) event. These are followed by a [Section 6.541](#) event which is generated by the parent PropertySheet and actually controls the state of the buttons. When the user clicks the Finish button, the PropertyPage generates a [Section 6.435](#) event alone. All of these events reported by the PropertyPage are reported for information only. Returning 0 from a callback function has no effect. You may however control the buttons using the [Section 6.541](#) event.

6.463 PropertySheet Object

Purpose: The PropertySheet object represents a standard multi-page dialog box.

Parents ActiveXControl Form OLEServer Root SubForm TCPSocket

Children Bitmap BrowseBox Circle Clipboard Cursor Ellipse FileBox Font Icon Locator Marker Metafile MsgBox Poly Printer PropertyPage Rect Text Timer TipField

Properties Type Caption Posn Size Style Coord Active Visible Event HasApply HasHelp PageActive PageActiveObject HelpButton FontObj OnTop Data EdgeStyle Handle Hint HintObj Tip TipObj Translate KeepOnClose MethodList ChildList EventList PropList

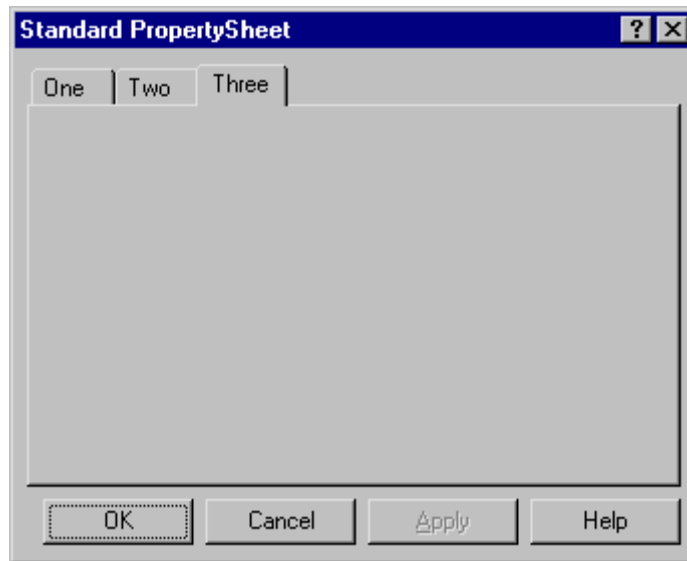
Methods Detach ChooseFont SetWizard SetFinishText CancelToClose GetFocus GetFocusObj Wait

Events Close Create FontOK FontCancel

Description

There are two different kinds of PropertySheet which you select using the Style property. This may only be set when the PropertySheet is created using `WC` and Style may not subsequently be changed using `WS`.

If Style is Standard (the default), the PropertySheet displays a set of pages (each represented by a [Section 6.462](#)) as a set of tabbed forms as illustrated below. The user selects the current page by clicking on the appropriate tab. This Style allows the user to select any page at any time and does not oblige the user to visit any but the first page you choose to display. This Style is useful for displaying groups of options or settings that the user may change.



If Style is Wizard, the PropertySheet displays its pages in succession starting with the first. The user steps from one to another using the Next and Back buttons and may be forced to visit all the pages in a prescribed order. This Style is useful for data entry or for asking the user to make a series of choices.



The Caption property specifies the text written in the window title bar, but only applies if the Style is Standard. The title bar text of a Wizard PropertySheet is specified by the Caption of the current [Section 6.462](#).

The HasApply and HasHelp properties are Boolean and specify whether or not the PropertySheet has "Apply" and "Help" buttons respectively. These properties may only be set when the object is created using `WC`. They both have default values of 1.

The FontObj and EdgeStyle properties have no effect on the appearance of the PropertySheet itself, but may be used to define the default appearance of its children.

6.464 PropList Property

Applies To

Description

This is a "read-only" property that supplies a list of all other properties which are applicable to the object in question. The list is returned as a vector of character vectors in the order in which the corresponding properties are expected by *Dyalog APL Language*: `Wc` and *Dyalog APL Language*: `Ws`.

EXAMPLE

```
'F'      WC 'Form'
'F.MB'   WC 'MenuBar'
'F.MB'   WG 'PropList'
Type Visible FontObj Data EdgeStyle MDIMenu PropList
```

6.465 Protected Event 470

Applies To: [Section 6.488](#)

Description

If enabled, this event is reported when the user attempts to alter protected text in a RichEdit. See [Section 6.87](#) property.

The event message reported as the result of `WM_QUERYOPEN`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Protected' or 470

6.466 QueueEvents Property

Applies To: [Section 6.412](#), [Section 6.414](#)

Description

The QueueEvents property specifies whether or not incoming events generated by an COM object are queued. It is a Boolean value where the (default) value 1 specifies that events are queued, and 0 that they are not.

If QueueEvents is 1, the result (if any) of your callback function is not passed back to the COM object but is discarded. Thus you cannot, for example, inhibit or modify the default processing of the event by the COM object.

If QueueEvents is 0, the following applies.

- The callback function attached to the event is executed immediately, even if there are other APL events before it in the internal event queue. This immediate execution means that your callback can fire during the execution of any other

function, including a callback function on an APL event. You must therefore take care that the callback makes no references to objects that may be shadowed.

- The result of your callback function is then passed back to the COM object. In this situation, it is essential that the callback is not interrupted by other events from the same, or another instance, of an COM object.
- To prevent APL itself from yielding to Windows, the Yield property is temporarily set to 0 while the callback is run. For the same reason, the tracing of a callback function, that is run immediately in this way, is disabled.

However, you must yourself also ensure that your own code does not yield. This means that you may not perform any operation in your callback that would yield to Windows; these include:

- `□DL`
- certain uses of `□NA`
- external function calls to Auxiliary Processors

If your callback does yield to Windows, thereby allowing another COM object event to arrive, this second event and any subsequent events that arrive during the execution of the callback are queued and will be processed later. These events may therefore not be modified by their callback functions.

6.467 Radius Property

Applies To: [Section 6.94](#), [Section 6.472](#)

Description

For a [Section 6.94](#) object, this property is a single number that specifies the radius of the circle/arc or a numeric vector that specifies the radii of a set of circles/arcs.

For a [Section 6.472](#) object, Radius is a 2-element vector that specifies the curvature of the corners of the rectangle or set of rectangles to be drawn. The curvature is defined in terms of the vertical and horizontal radii of an ellipse. The first element of Radius defines the radius vertically, the second horizontally. If more than one rectangle is involved, either or both of the elements of Radius may be vectors. The default value is (0,0) which gives square corners.

6.468 RadiusMode Property

Applies To: [Section 6.94](#), [Section 6.489](#)

Description

A perfectly round circle can only be drawn if the diameter is an odd number of pixels. The RadiusMode property specifies whether or not a circle is adjusted by a single pixel, if necessary, so as to appear perfectly round.

If RadiusMode is 1 or -1, and the diameter is an even number of pixels, the circle is actually drawn with a diameter of 1 pixel more or less than specified. If RadiusMode is 0 (the default), no such adjustment is made.

RadiusMode may be set on the Root object to be inherited by all Circle objects.

6.469 Range Property

Applies To: [Section 6.227](#), [Section 6.508](#), [Section 6.578](#)

Description

This property determines the maximum value of the thumb in a scrollbar (the minimum value is always 1). This may be any positive integer value that is greater than 1.

For a [Section 6.508](#) object Range is a single number. For a [Section 6.227](#) or [Section 6.578](#) object, Range is a 2-element vector which specifies the maxima for the [Section 6.227](#)'s vertical and horizontal scrollbars respectively.

6.470 ReadOnly Property

Applies To: [Section 6.55](#), [Section 6.56](#), [Section 6.185](#), [Section 6.563](#)

Description

This property specifies whether or not the user may alter the text in an object. The default value of ReadOnly is 0 which allows the user to alter text.

If you set ReadOnly to 1, a cursor is displayed in the object, the user may navigate around the text in the usual manner with the mouse and/or the keyboard and select

text and copy it to the clipboard. However, all input that would otherwise change the data is ignored.

For a Button object with [Section 6.577](#) 'Radio' or 'Check', setting ReadOnly to 1 prevents the user from changing the state of the [Section 6.55](#), although mouse and other events will still be reported.

6.471 RealSize Property

Applies To: [Section 6.385](#)

Description

There are several distinct types of Windows metafiles. A *placeable* metafile is one that carries with it its *suggested size*. Certain programs (such as Word for Windows) only support placeable metafiles.

The RealSize property specifies the suggested size of a Metafile in units of 0.01mm. Thus to make a placeable Metafile with a suggested size of 20 x 10 cm, you would set RealSize to (20000 10000).

The RealSize property is not used or required by Dyalog and is provided only to enable you to make and save a new metafile that is placeable. If you create a Metafile object from a file, the value of RealSize will be obtained from the value recorded in the file (if it is placeable). Otherwise, RealSize will be (0 0). If so, you must set RealSize to make it placeable. Each element of RealSize must be an integer in the range 0-144745.

6.472 Rect Object

Purpose: A graphical object used to draw boxes.

Parents ActiveXControl Animation Bitmap Button ButtonEdit Combo ComboEx Edit Form Grid Group Label List ListView MDIClient Metafile Printer ProgressBar PropertyPage PropertySheet RichEdit Scroll Spinner Static StatusBar SubForm TabBar TipField ToolBar TrackBar TreeView UpDown

Children Timer

Properties Type Points Size Radius FCol BCol LStyle LWidth FStyle FillCol Coord Visible Event Dragable OnTop CursorObj AutoConf Data EdgeStyle Accelerator KeepOnClose DrawMode MethodList ChildList EventList PropList

Methods Detach

Events Close Create DragDrop MouseDown MouseUp MouseMove MouseDbIClick Help Select

Description

The [Section 6.449](#) property specifies one or more sets of co-ordinates which define the position(s) of one or more rectangles. The position of a rectangle is defined to be the position of the corner that is **nearest** to the origin of its parent. The default is therefore its top-left corner. The [Section 6.556](#) property specifies the height and width of each rectangle, measuring away from the origin.

The [Section 6.467](#) property specifies the curvature of the corners of the rectangle.

[Section 6.360](#) and [Section 6.361](#) define the style and width of the lines used to draw the boundaries of the rectangle(s). [Section 6.205](#) and [Section 6.45](#) determine the colour of the lines.

[Section 6.231](#) specifies whether or not the rectangle(s) are filled, and if so, how. For a solid fill ([Section 6.231](#) 0), [Section 6.214](#) defines the fill colour used. For a pattern fill ([Section 6.231](#) 1-6) [Section 6.214](#) defines the colour of the hatch lines and [Section 6.45](#) the colour of the spaces between them.

The [Section 6.184](#) property may specify a 3-dimensional effect. If so, the boundary line around the rectangle is replaced by a border designed to achieve the desired effect.

The value of [Section 6.173](#) determines whether or not the object can be dragged. The value of [Section 6.35](#) determines whether or not the Rect object is resized when its parent is resized.

The structure of the property values is best considered separately for single and multiple rectangles :

Single Rectangle

For a single rectangle, [Section 6.449](#) is either a 2-column matrix of (y,x) co-ordinates, or a 2-element vector of y and x co-ordinates respectively.

[Section 6.556](#) is a simple 2-element vector whose elements specify the height and width of the rectangle respectively.

[Section 6.467](#) is a 2-element vector which specifies the major (y-axis) and minor (x-axis) radii of an ellipse used to draw the corners of the rectangle. Its default value is (0 0) which yields right-angled corners.

[Section 6.360](#) and [Section 6.361](#) are both simple scalar numbers.

[Section 6.231](#) is either a single number specifying a standard fill pattern, or the name of a [Section 6.47](#) object which is to be used as a "brush" to fill the rectangle.

[Section 6.205](#), [Section 6.45](#) and [Section 6.214](#) are each either single numbers representing standard colours, or 3-element vectors which specify colours explicitly in terms of their RGB values.

EXAMPLES

First make a [Section 6.227](#) :

```
'F' □WC 'Form'
```

Draw a single rectangle at (y=10, x=5) with height=30, width=50 :

```
'F.R1' □WC 'Rect' (10 5)(30 50)
```

Ditto with rounded corners (radii 10) :

```
'F.R1' □WC RC←'Rect' (10 5)(30 50)(10 10)
```

Ditto, but use a red line :

```
'F.R1' □WC RC,(255 0 0)
```

Ditto, but fill in green

```
'F.R1' □WC RC(255 0 0) ('FStyle' 0)(0 255 0)
```

Multiple Rectangles

To draw a set of rectangles with a single name, [Section 6.449](#) may be a simple 2-element vector (specifying the location of all the rectangles), **or** a 2-column matrix whose first column specifies their y-coordinates and whose second column specifies their x-coordinates, **or** a 2-element nested vector whose first element specifies their y-coordinate(s) and whose second element specifies their x-coordinate(s).

Likewise, [Section 6.556](#) may be a simple 2-element vector (applying to all the rectangles), **or** a 2-column matrix whose first column specifies their heights and whose second column specifies their widths, **or** a 2-element nested vector whose first element specifies their height(s) and whose second element specifies their width(s).

Section 6.467 may be a simple 2-element vector (applying to all the rectangles), **or** a 2-column matrix whose first column specifies major radii and whose second column specifies minor radii, **or** a 2-element nested vector whose first element specifies major radii and whose second element specifies minor radii.

Section 6.360 and Section 6.361 may each be simple scalar values (applying to all the rectangles) or simple vectors whose elements refer to each of the corresponding rectangles in turn.

Section 6.231 may be a simple scalar numeric or a simple character vector (Section 6.47 name) applying to all rectangles, or a vector whose elements refer to each of the corresponding rectangles in turn.

Similarly, Section 6.205, Section 6.45 and Section 6.214 may each be single numbers or a single (enclosed) 3-element vector applying to all the rectangles. Alternatively, these properties may contain vectors whose elements refer to each of the rectangles in turn. If so, their elements may be single numbers or nested RGB triplets, or a combination of the two.

EXAMPLES

First make a Section 6.227 :

```
'F' □WC 'Form'
```

Draw two rectangles at (y=5, x=10) and (y=5, x=60) each of (height=40, width=10)

```
'F.R1' □WC 'Rect' ((5 5)(10 60)) (40 10)
```

Ditto, using scalar extension for (y=5) :

```
'F.R1' □WC 'Rect' (5(10 60)) (40 10)
```

Ditto, but draw the first with (height=40, width=30) and the second with (height=20, width=10) :

```
'F.R1' □WC 'Rect' (5(10 60)) ((40 20)(30 10))
```

Draw two rectangles at (y=5, x=10) and (y=5, x=60) each of (height=40, width=10) and with rounded corners of radii (10,10) :

```
'F.R1' □WC RC←'Rect' (5(10 60)) (40 10) (10 10)
```

Ditto, using a green line for both :

```
'F.R1' □WC RC, <=0 255 0
```

Ditto, but using red and blue lines respectively :

```
'F.R1' □WC RC, <(255 0 0)(0 0 255)
```

6.473 Redraw Property

Applies To

Description

The Redraw property specifies whether or not APL automatically redraws an object when it is exposed or when any of its properties change in a way that would affect its appearance.

The value reported by the Redraw property is a Boolean value; 1 means that APL automatically redraws the object when necessary (the default); 0 means that APL does not redraw the object.

Setting Redraw to 0 or 1 affects only whether or not APL will redraw the object from then on.

Setting Redraw to 2 has the same effect as setting it to 1, but the object is also redrawn immediately. Note that no child object is redrawn.

Setting Redraw to 3 has the same effect as setting it to 1, but the object and its children are redrawn immediately.

6.474 RemoteAddr Property

Applies To: [Section 6.604](#)

Description

The RemoteAddr property is a character vector that specifies the IP address of the remote computer.

RemoteAddr may only be specified by a client [Section 6.604](#) that is intended to make a connection with a server. Furthermore, it must be specified in the □WC statement that creates the [Section 6.604](#) object and it may not subsequently be changed using □WS.

You may use either RemoteAddr or [Section 6.475](#) to identify the remote computer. If you know its IP address, it is normally quicker to specify RemoteAddr. If you specify both properties, the value of [Section 6.475](#) will be ignored.

For a server [Section 6.604](#), RemoteAddr is determined by the IP address of the connecting process and is a read-only property.

6.475 RemoteAddrName Property

Applies To: [Section 6.604](#)

Description

The RemoteAddrName property is a character vector that specifies the host name of the remote computer to which you wish to make a connection.

RemoteAddrName may only be specified by a client [Section 6.604](#) that is intended to make a connection with a server. Furthermore, it must be specified in the `□wc` statement that creates the [Section 6.604](#) object and it may not subsequently be changed using `□ws`.

When the specified host name has been resolved to an IP address, the [Section 6.604](#) will generate a [Section 6.598](#) event and update the value of [Section 6.474](#) accordingly.

Note that you may use *either* [Section 6.474](#) or RemoteAddrName to identify the remote computer. If you know its IP address, it is normally quicker to specify [Section 6.474](#). If you specify both properties, the value of RemoteAddrName will be ignored.

For a server [Section 6.604](#), you may not specify RemoteAddrName and `□wg` returns an empty character vector.

6.476 RemotePort Property

Applies To: [Section 6.604](#)

Description

The RemotePort property is a scalar integer in the range 1-65536 that identifies the [Section 8.7](#) associated with a service on a remote computer.

RemotePort may only be specified by a client [Section 6.604](#) that is intended to make a connection with a server. Furthermore, it must be specified in the `□wc` statement that creates the [Section 6.604](#) object and it may not subsequently be changed using `□ws`.

Note that you may use *either* RemotePort or [Section 6.477](#) to identify the remote service. If you know the port number, it is normally quicker to specify RemotePort. However unless it is a *well known port number*, the use of a port name is generally more flexible. If you specify both properties, the value of [Section 6.477](#) will be ignored.

For a server [Section 6.604](#), RemotePort is determined by the [Section 8.7](#) of the connecting process and is a read-only property.

6.477 RemotePortName Property

Applies To: [Section 6.604](#)

Description

The RemotePortName property is a character vector that specifies the port name of the remote service to which you wish to make a connection.

RemotePortName may only be specified by a client [Section 6.604](#) that is intended to make a connection with a server. Furthermore, it must be specified in the `□wc` statement that creates the [Section 6.604](#) object and it may not subsequently be changed using `□ws`.

When the specified port name has been resolved to a port number, the [Section 6.604](#) will generate a [Section 6.599](#) event and update the value of [Section 6.476](#) accordingly.

Note that you may use *either* [Section 6.476](#) or RemotePortName to identify the remote service. If you know the port number, it is normally quicker to specify [Section 6.476](#). However unless it is a *well known port number*, the use of a port name is generally more flexible. If you specify both properties, the value of RemotePortName will be ignored.

For a server [Section 6.604](#), you may not specify RemotePortName and `□wg` returns an empty character vector.

6.478 ReportBCol Property

Applies To: [Section 6.348](#)

Description

In Report View, the ReportBCol property is either a scalar or a matrix that specifies the background colours for each item displayed in a [Section 6.348](#) object .

Its first column refers to the [Section 6.334](#) themselves, and subsequent columns to the elements of [Section 6.480](#).

That is, if non-scalar, $(\text{pReportBCol}) \leftrightarrow (0 \ 1 + \text{pReportInfo})$

Each element of ReportBCol is either an integer colour value or a 3-element of RGB colour indices.

For further information, see [Section 6.45](#).

6.479 ReportImageIndex Property

Applies To: [Section 6.348](#)

Description

The ReportImageIndex property is an integer scalar or matrix that specifies the images to be displayed alongside each item in a [Section 6.348](#) object in Report View.

If it is a matrix, its first column specifies the indices of the icons to be displayed against the [Section 6.334](#) of the [Section 6.348](#), overriding the icons specified by [Section 6.315](#), and its subsequent columns specify the indices of the icons to be displayed against the elements of [Section 6.480](#).

That is, if non-scalar, $(\text{pReportImageIndex}) \leftrightarrow (0 \ 1 + \text{pReportInfo})$

Each element of ReportImageIndex specifies an index into the [Section 6.316](#) object specified by the [Section 6.317](#) property.

6.480 ReportInfo Property

Applies To: [Section 6.348](#)

Description

The ReportInfo property is a matrix that is displayed alongside each item in a [Section 6.348](#) object when its [Section 6.644](#) property is 'Report'. Each element of the matrix may be a character vector or a number.

The information is displayed in a grid format, the first column of which contains the item labels and their icons. Subsequent columns of the grid are defined by the corresponding columns of ReportInfo. The alignment of the columns is specified by the [Section 6.114](#) property.

6.481 ResizeCols Property

Applies To: [Section 6.263](#)

Description

This property determines whether or not the user may resize columns in the [Section 6.263](#). It is a Boolean scalar or vector with one element per column. A value of 1 indicates that the corresponding column is resizable by the user. A value of 0 means that the corresponding column may not be resized by the user.

If a column is resizable, the cursor changes to a double headed arrow when the mouse pointer is placed over the right-hand border of the column title. The user may resize the column by dragging this border. The user may also resize a column by double-clicking over its right-hand border. This causes the column to be resized to fit the data and the width of the column is automatically adjusted to display the widest value in any of its cells. Either operation generates a [Section 6.527](#) event.

Note that the user may cause the column to disappear altogether by dragging it to a zero width. Once this has been done, this column may only be restored if the column to its left is itself not resizable.

6.482 ResizeColTitles Property

Applies To: [Section 6.263](#)

Description

This property determines whether or not the user may alter the height of the column titles in the [Section 6.263](#). It is either 1, which indicates that the height of the column titles is adjustable by the user, or 0 which means that it is not.

If the height of the column titles is adjustable, the cursor changes to a double headed arrow when the mouse pointer is placed over the top border of the first row title. The user may resize the column titles by dragging this border. The user may also resize the column titles by double-clicking over this border. This causes the column titles to be resized to fit the data and the height of the column titles is automatically adjusted to display the tallest heading in any of its columns. Either operation generates a [Section 6.536](#) event. The value of the row number reported by the event is -1 .

Note that the user may cause the column titles to disappear altogether by dragging them to a zero height. Once this has been done, the row titles cannot be restored.

6.483 ResizeRows Property

Applies To: [Section 6.263](#)

Description

This property determines whether or not the user may resize rows in the [Section 6.263](#). It is a Boolean scalar or vector with one element per column. A value of 1 indicates that the corresponding row is resizable by the user. A value of 0 means that the corresponding row may not be resized by the user.

If a row is resizable, the cursor changes to a double headed arrow when the mouse pointer is placed over the lower border of the row title. The user may change the height of the row by dragging this border up and down. The user may also resize a row by double-clicking over its bottom border. This causes the row to be resized to fit the data and the height of the row is automatically adjusted to display the tallest value in any of its cells. Either operation generates a [Section 6.536](#) event.

Note that the user may cause the row to disappear altogether by dragging it to a zero height. Once this has been done, this row may only be restored if the row above it is itself not resizable.

6.484 ResizeRowTitles Property

Applies To: [Section 6.263](#)

Description

This property determines whether or not the user may alter the width of the row titles in the [Section 6.263](#). It is either 1, which indicates that the width of the row titles is adjustable by the user, or 0 which means that it is not.

If the width of the row titles is adjustable, the cursor changes to a double headed arrow when the mouse pointer is placed over the left-hand border of the first column title. The user may resize the row titles by dragging this border. The user may also resize the row titles by double-clicking over this border. This causes the row titles to be resized to fit the data and the width of the row titles is automatically adjusted to display the longest string in any of its rows. Either operation generates a [Section 6.527](#) event. The value of the column number reported by the event is -1 .

Note that the user may cause the row titles to disappear altogether by dragging them to a zero width. Once this has been done, the row titles cannot be restored.

6.485 Resolution Property

Applies To: [Section 6.455](#)

Description

The Resolution property determines the print resolution.

You may set Resolution to 'Draft', 'Low', 'Medium' or 'High'.

Alternatively, you can set Resolution to a 2-element integer vector that specifies the desired number of dots per inch in the x (horizontal) and y (vertical) direction respectively.

The initial value reported by Resolution may be reported in either form (character vector or 2-element numeric vector) according to the current printer settings.

6.486 Resolutions Property

Applies To: [Section 6.455](#)

Description

The Resolutions property is a read-only property that reports the available printer resolutions.

Resolutions is a vector of 2-element integer vectors each of which specifies the number of dots per inch in the x (horizontal) and y (vertical) directions respectively.

6.487 Retracting Event 304

Applies To: [Section 6.263](#), [Section 6.629](#)

Description

If enabled, this event is reported by a [Section 6.263](#) or a [Section 6.629](#) object just before it is about to retract to hide the children of the current item.

In a [Section 6.263](#), this occurs when the user clicks the picture or tree line in the row title.

In a [Section 6.629](#), this occurs when the user double-clicks the item label or clicks in the button or on the tree line to the left of the item label, when the item is in its expanded state.

The default processing for the event is to retract the tree at the corresponding point.

You may disable the retract operation by setting the action code for the event to -1. You may also prevent the retraction from occurring by returning 0 from a callback function. You may retract [Section 6.263](#) a or a [Section 6.629](#) dynamically under program control by calling Retracting as a method.

The event message reported as the result of □□□, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Retracting' or 304
[3]	Item number	Integer. The index of the item.

6.488 RichEdit Object

Purpose: The RichEdit object is a multi-line text editor that provides a wide range of word-processing capabilities.

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm ToolBar ToolControl

Children Bitmap Circle Cursor Ellipse Font Icon Marker Poly Rect Text Timer

Properties Type Text Posn Size File Coord Border Active Visible Event VScroll HScroll SelText Sizeable Dragable FontObj FCol BCol CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Changed RTFText Translate Accelerator CharFormat WordFormat ParaFormat PageWidth AcceptFiles WantsReturn KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj RTFPrintSetup RTFPrint FileRead FileWrite

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap Protected Change Select

Description

A RichEdit object is a window in which the user can enter and edit text. The text can be assigned character and paragraph formatting. It is implemented using the Microsoft Windows RichEdit Version 1.0 control, although embedded OLE objects are not supported.

The RichEdit object provides a programming interface for formatting text. However, your application must implement any user interface components necessary to make formatting operations available to the user. For example, your program can set the colour and font of a particular block of text, but the RichEdit itself provides no facilities for the user to do this directly. It is up to you to provide these.

The [Section 6.207](#) property specifies the name of a file associated with the object. Data in the file is assumed to be in rich text format, and the default extension for the file is .RTF. You can read the file into the object by calling [Section 6.212](#) and you can write the contents to the file by calling [Section 6.213](#). You can also print the contents of the object by calling [Section 6.504](#).

The [Section 6.606](#) property may be used to set or retrieve the text of the RichEdit, but ignores formatting information. [Section 6.606](#) may set using a simple character vector, a simple matrix, or a vector of vectors. If [Section 6.606](#) is specified by a matrix or by a vector of vectors, "new-line" characters are automatically added at the end of each line in the RichEdit control.

The user may insert a "new-line" character in the text by pressing Ctrl-Enter. If [Section 6.606](#) was set by a matrix, it is returned as a matrix. Otherwise it is returned as a vector of vectors. "New-line" characters are not returned. If [Section 6.606](#) was not specified by *Dyalog APL Language: Wc* or *Dyalog APL Language: Wsit* it is returned an empty matrix (1 0ρ' ').

The [Section 6.506](#) property may be used to set or retrieve the contents of the RichEdit, including text and formatting.

The [Section 6.439](#) property defines the width of the text within the object. Text entered into the object is automatically wrapped according to [Section 6.439](#). This property also defines the width when the text is printed.

You can set the default character format or the format of a particular block of text using the [Section 6.87](#) property. If there is no selection, setting [Section 6.87](#) defines the default character format that applies at the current insertion position and establishes the appearance of all of the text (font, colour, size etc.) that the user subsequently enters here. If there is a selection, setting [Section 6.87](#) sets the character format for the selected block of text.

The [Section 6.659](#) property is similar to [Section 6.87](#) except that it sets the format for the selected word(s) or, if there is no selection, for the word containing the insertion point.

The [Section 6.444](#) property defines the paragraph formatting which includes alignment, indentation and the location of tab stops. When you set [Section 6.444](#) with ¶ws, the formatting is applied to the current selection. If there is no selection, it defines the default paragraph formatting at the insertion point.

All of the dimensions used for text and paragraph formatting are specified in [Section 8.9](#). You can convert from pixels to [Section 8.9](#) and vice versa using the DevCaps property of either Root or the Printer object as appropriate.

The behaviour of the Enter key is defined by the [Section 6.650](#) property. If [Section 6.650](#) is 1 (the default), the Enter key inputs a new line into the RichEdit object. If [Section 6.650](#) is 0 the Enter key is ignored by the RichEdit object and may instead generate a Select event on a Button. In this case the user must press Ctrl+Enter to input a new line.

The user may copy and paste information (in RTF format) between a RichEdit object and the Windows clipboard. The Clipboard object also has an [Section 6.506](#) property that supports RTF format

If the user attempts to alter text that is protected (see [Section 6.87](#)) the RichEdit object reports a [Section 6.465](#) event.

You may print the contents of a RichEdit object using the [Section 6.504](#) method. You may display a print set-up dialog box using the [Section 6.505](#) method.

6.489 Root Object

Purpose: This is an invisible "system" object that acts as the parent of all other objects.

Children Bitmap BrowseBox Clipboard Cursor FileBox Font Form HTMLRenderer Icon ImageList Locator Menu Metafile MsgBox NetClient NetType OCXClass OLEClient OLEServer Printer PropertySheet SysTrayItem TCPSocket Timer TipField

Properties Type Caption Posn Size DevCaps Coord Event FontObj FontList PrintList IconObj CursorObj YRange XRange Data TextSize Yield EdgeStyle HintObj TipObj Translate APLVersion EvaluationDays KeepOnClose OLEControls OLEServers LastError RadiusMode MethodList ChildList EventList PropList

Methods ChooseFont Flush NameFromHandle GreetBitmap GetCommandLine GetCommandLineArgs GetEnvironment ListTypeLibs DeleteTypeLib SetServiceState

GetServiceState GetTextSize GetFocus ShowSIP GetFocusObj IDNToDate DateToIDN
TCPGetHostID GetBuildID Wait

Events FontOK FontCancel DDE Idle ExitWindows ExitApp WinIniChange
SysColorChange DisplayChange ActivateApp ServiceNotification

Description

There is a single Root object called ' .' which is always present. It cannot be created using *Dyalog APL Language: Wc* nor can it be destroyed.

The [Section 6.66](#) and [Section 6.310](#) properties of ' .' are used to identify a Dyalog APL/W application as distinct from the APL Session. The [Section 6.66](#) property specifies the application name that is displayed when you cycle through running applications using Alt+Tab and by the Windows Task List. The [Section 6.310](#) property specifies the name of an [Section 6.309](#) object that is displayed alongside the application name in the box displayed by Alt+Tab. For these to take effect, your application must have at least one visible and active [Section 6.227](#).

For the Root object, the value of [Section 6.452](#) is (0,0). The value of [Section 6.556](#) is either (100,100) if [Section 6.128](#) is 'Prop', or the size of the screen in pixels if [Section 6.128](#) is 'Pixel'. [Section 6.661](#) and [Section 6.663](#) both have the value (0,100). The [Section 6.158](#) property reports the physical size of the screen in terms of both pixels and millimetres. It also reports the number of colours available.

The [Section 6.224](#) property provides a list of all the character fonts that are available. The [Section 6.456](#) property provides a list of all the installed printers. These properties are *read-only* and may not be changed using *Dyalog APL Language: Ws*

As the default value of [Section 6.128](#) is 'Inherit' for all other objects, the value of [Section 6.128](#) for ' .' defines the default co-ordinate system. It may be either 'Prop' (the default) or 'Pixel'. 'Inherit' and 'User' are not allowed.

The [Section 6.137](#) property is used to define a cursor for the application as a whole. Its default value is an empty character vector. If it is set to any value other than ' ' or 0, the selected cursor overrides the [Section 6.137](#) values for all other objects. If you want to indicate that the application is "busy", you can therefore set the [Section 6.137](#) property on ' .' to an hourglass for the duration of the operation. For example:

```
'.' ⓂWS 'CursorObj' 1  Ⓜ Set cursor to an hourglass
```

[lengthy process...]

```
'.' ⓂWS 'CursorObj' 0  Ⓜ Reset cursor
```

The [Section 6.662](#) property specifies how frequently APL yields to Windows during the execution of code. Its default value is 200 milliseconds.

The [Section 6.184](#) property is used to determine whether or not objects may have 3-dimensional effects. Setting [Section 6.184](#) to 'None' disables 3-dimensional effects on all [Section 6.227s](#) and controls. Setting [Section 6.184](#) to any other value enables 3-dimensional effects for these objects.

The [Section 6.199](#) and [Section 6.200](#) events can be used to prevent the user closing your application from the Windows Task List or by terminating Windows.

The expression `⊖EX ' '` deletes all objects owned by the current thread **except** the Root object itself. In addition, if this expression is executed by thread 0, it resets all the properties of ' ' to their default values.

Exposing Root members

The Properties, Methods and Events of the Root object are always accessible using the system functions `⊖WS`, `⊖WG` and `⊖NQ` but may also be optionally accessed directly as if they were global variables or functions in the workspace.

For example, if Root members are exposed, the following expression will set the application cursor (GUI) to an hourglass:

```
CursorObj←1
```

There are a number of elements that control whether or not Root members are exposed.

1. The fundamental mechanism is a flag that is saved in every workspace. If this flag is set, the members of the Root object are exposed; if not, they are not exposed.
2. This flag may be changed dynamically using the *Options/Object Syntax/Expose Root Properties* menu item on the Session or using `2401±`. If the workspace is subsequently saved, the current value of the flag is saved with it.
3. The value of the flag in a `CLEAR WS` is determined by the **PropertyExposeRoot** parameter. Under Windows, this parameter is associated with the *Expose properties of Root* checkbox on the *Object Syntax* Tab of the *Configuration* dialog box. When you change the value of this checkbox and close the *Configuration* dialog by clicking OK, the value of the **PropertyExposeRoot** parameter is immediately updated in the user's section of the Registry. However, the value of the flag *in the current workspace* is not changed. The **PropertyExposeRoot** parameter only defines the value of the flag in a `CLEAR WS`, so if you

subsequently type `)CLEAR`, the current value of the parameter in the Registry determines whether or not Root members are exposed and sets the flag in the workspace accordingly.

For further information, see *User Interface: Session Menubar, Installation/Configuration: Configuration Parameters*, and *Dyalog APL Language: Expose Root Properties*.

Notes

1. When Root members are exposed, the first reference or assignment to a member, associates that name (with a nameclass of `~2.6` or `~3.6`) with that member. If, having referenced a member in this way, you subsequently hide Root members using `(2401±0)`, that name remains connected to that member and the member remains exposed. This association may however be removed by erasing the name.
2. If Root members are not exposed, you are free to define an APL object with the same name as one of the members. If you subsequently expose Root members using `(2401±1)`, the name remains associated with the APL object and not with a member of Root. If you then erase the name and re-reference or re-assign it, the name will be associated with the corresponding member.

6.490 Rotate Property

Applies To: [Section 6.222](#)

Description

This property specifies the angle of rotation of the font measured in radians ($0 \rightarrow \infty$) from the x-axis in a counter-clockwise direction. Note that only TrueType fonts can be rotated. Rotated fonts are supported **only** for use with the [Section 6.605](#) object.

6.491 RowChange Method 158

Applies To: [Section 6.263](#)

Description

This method is used to change the data in a row of a [Section 6.263](#) object.

The argument to RowChange is a 2-element array as follows.

[1]	Row number	integer
[2]	Row data	array

Row data must be a scalar or a vector whose length is equal to the number of columns in the Grid. Its elements may be scalar numbers, character vectors or matrices.

6.492 RowHiddenDepth Property

Applies To: [Section 6.263](#)

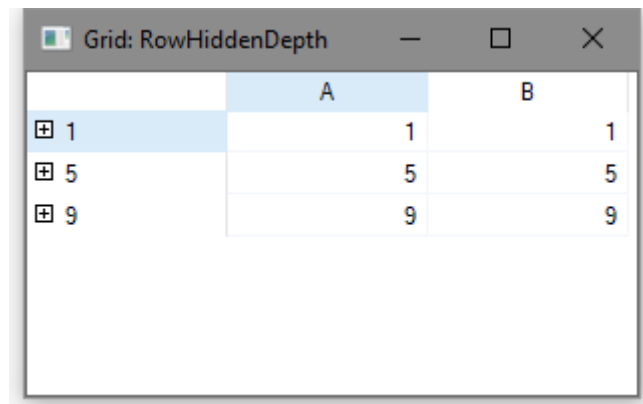
Description

The RowHiddenDepth property is read-only and identifies which rows of a Grid are currently hidden.

RowHiddenDepth is an integer vector with the same number of elements as there are rows in the Grid. The values in RowHiddenDepth indicate the current depth of the row in the visible hierarchy; that is, number of nodes that must be opened to display it. The value 0 means that the corresponding row is visible. The value 1 means that 1 node must be opened to display it; 2 means 2 nodes, and so forth.

EXAMPLE

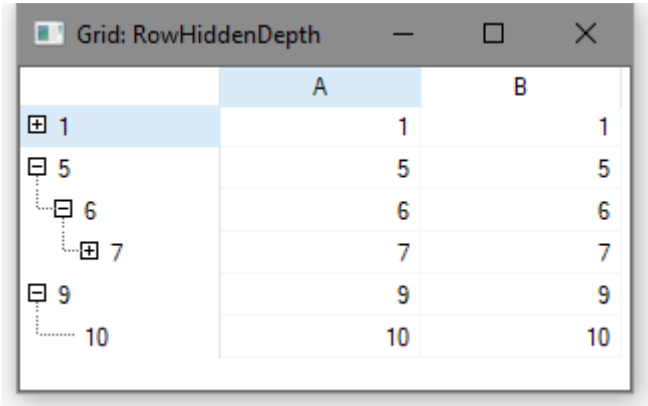
```
'F'WC'Form' 'Grid: RowHiddenDepth'
F.(Coord Size)←'Pixel'(163 306)
'F.G'WC'Grid'(10 2p2/110)('Size'F.Size)
←F.G.RowTreeDepth←10p0 1 2 3
0 1 2 3 0 1 2 3 0 1
```



With all nodes closed, RowHiddenDepth is the same as RowTreeDepth.

```
F.G.RowHiddenDepth
0 1 2 3 0 1 2 3 0 1
```

The next picture shows the Grid after the user has opened nodes 5,6 and 9.



```
F.G.RowHiddenDepth
0 1 2 3 0 0 0 1 0 0
```

6.493 RowLineTypes Property

Applies To: [Section 6.263](#)

Description

This property specifies the appearance of the horizontal grid lines in a [Section 6.263](#) object.

RowLineTypes is an integer vector, whose length is normally equal to the number of rows in the [Section 6.263](#). Each element in RowLineTypes specifies an index into the [Section 6.272](#) and [Section 6.274](#) properties, thus selecting the colour and width of the horizontal grid lines.

For example, if RowLineTypes[1] is 3, the first horizontal grid line in the [Section 6.263](#) is displayed using the colour specified by the 3rd element of [Section 6.272](#), and the width specified by the 3rd element of [Section 6.274](#).

Note that RowLineTypes is not $\square$ IO dependent, and the value 0 is treated the same as the value 1; both selecting the *first* colour and line width specified by [Section 6.272](#) and [Section 6.274](#) respectively.

The default value of RowLineTypes is an empty numeric vector ($\emptyset$). If so, all horizontal grid lines are drawn using the first element of [Section 6.272](#) and [Section 6.274](#).

A horizontal grid line is drawn along the bottom edge of its associated row. One pixel is drawn *inside* the row of cells; additional pixels (if any) are drawn *between* that row of cells and the next one below.

6.494 Rows Property

Applies To: [Section 6.121](#), [Section 6.122](#)

Description

For [Section 6.121](#) objects with [Section 6.577](#) 'Drop' or 'DropEdit' this property determines the number of rows displayed in the drop-down listbox when it is displayed. Note that the height of the edit field of a [Section 6.121](#) of this type is dependent only upon the size of the font in use, and cannot otherwise be changed.

Rows is a "read-only" property for a [Section 6.121](#) with [Section 6.577](#) 'Simple' and an attempt to set it in a [Section 6.121](#) of this type with *Dyalog APL Language: Wc* or *Dyalog APL Language: Ws* will generate a NONCE ERROR. Instead, the overall height of a Simple [Section 6.121](#) is determined by the first element of the [Section 6.556](#) property.

6.495 RowSetVisibleDepth Method 173

Applies To: [Section 6.263](#)

Description

This method is used to set the maximum visible depth of data in rows of a [Section 6.263](#).

The argument to RowSetVisibleDepth is a numeric scalar as follows

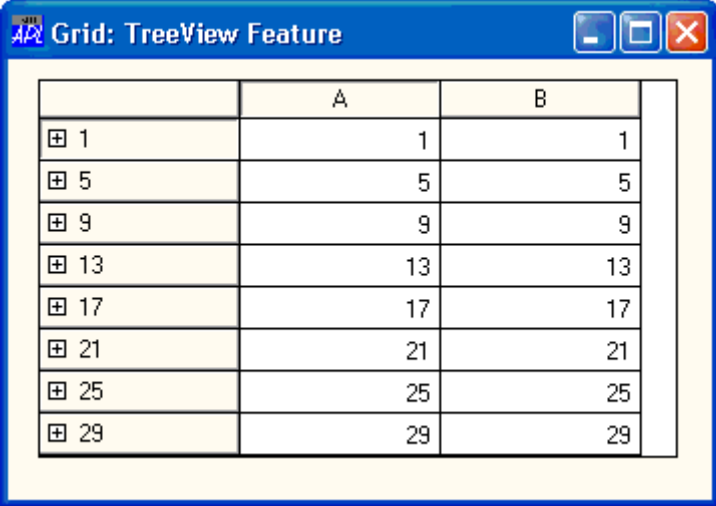
[1]	Depth	integer

All rows in the grid that have a value of [Section 6.501](#) less than or equal to *Depth* are expanded. Rows with a value of [Section 6.501](#) greater than *Depth* are collapsed.

Note: [Section 6.201](#) and [Section 6.487](#) events are not generated when this method is called.

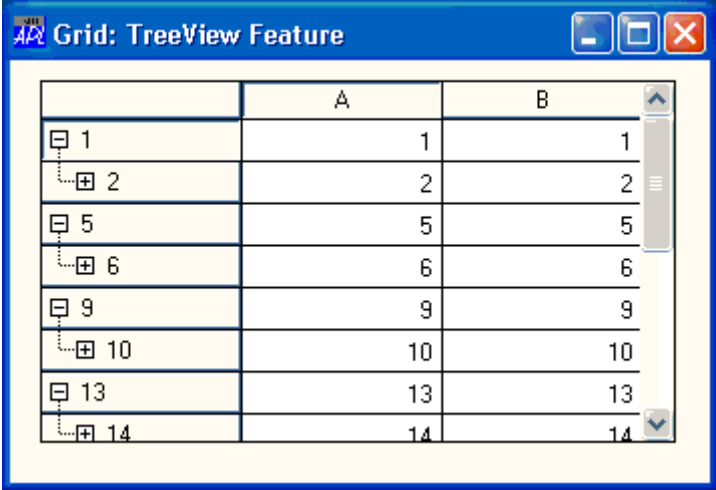
EXAMPLES

```
'F'WC'Form' 'Grid: TreeView Feature'
'F.G'WC'Grid'(30 2p2/130)
F.G.RowTreeDepth←30p0 1 2 2
```



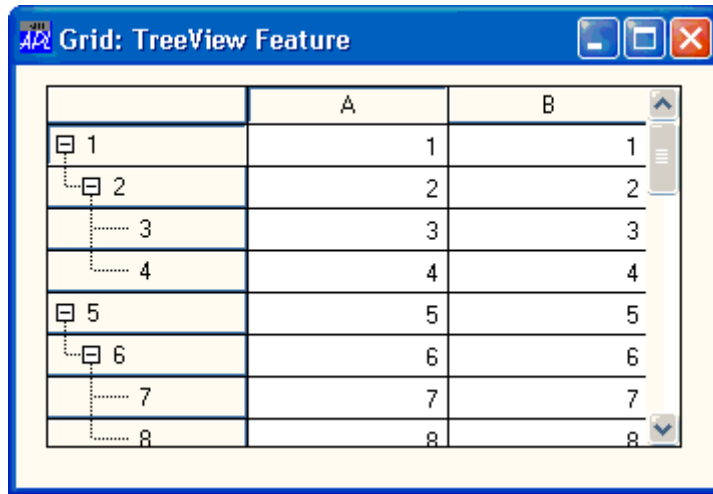
	A	B
⊕ 1	1	1
⊕ 5	5	5
⊕ 9	9	9
⊕ 13	13	13
⊕ 17	17	17
⊕ 21	21	21
⊕ 25	25	25
⊕ 29	29	29

```
F.G.RowSetVisibleDepth 1
```



	A	B
⊖ 1	1	1
⋮ ⊕ 2	2	2
⊖ 5	5	5
⋮ ⊕ 6	6	6
⊖ 9	9	9
⋮ ⊕ 10	10	10
⊖ 13	13	13
⋮ ⊕ 14	14	14

```
F.G.RowSetVisibleDepth 99
```



6.496 RowTitleAlign Property

Applies To: [Section 6.263](#)

Description

The RowTitleAlign property specifies the alignment of row titles in a Grid. It is either a simple character vector, or a vector of character vectors with one element per row.

An element of RowTitleAlign may be: 'Top', 'Bottom', 'Left', 'Right', 'Centre', 'TopLeft', 'TopRight', 'BottomLeft', or 'BottomRight'.

Note that both spellings 'Centre' and 'Center' are accepted.

6.497 RowTitleBCol Property

Applies To: [Section 6.263](#)

Description

The RowTitleBCol property specifies the background colour of the row titles in a [Section 6.263](#) object

RowTitleBCol may be a scalar that specifies a single background colour to be used for all of the row titles, or a vector that specifies the background colour of each of the row titles individually. An element of RowTitleBCol may be an enclosed 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of

the colour respectively, or it may be a scalar that defines a standard Windows colour element (see [Section 6.45](#) for details). Its default value is 0 which obtains the colour defined for Button Face.

6.498 RowTitleDepth Property

Applies To: [Section 6.263](#)

Description

RowTitleDepth specifies the structure of a set of hierarchical row titles.

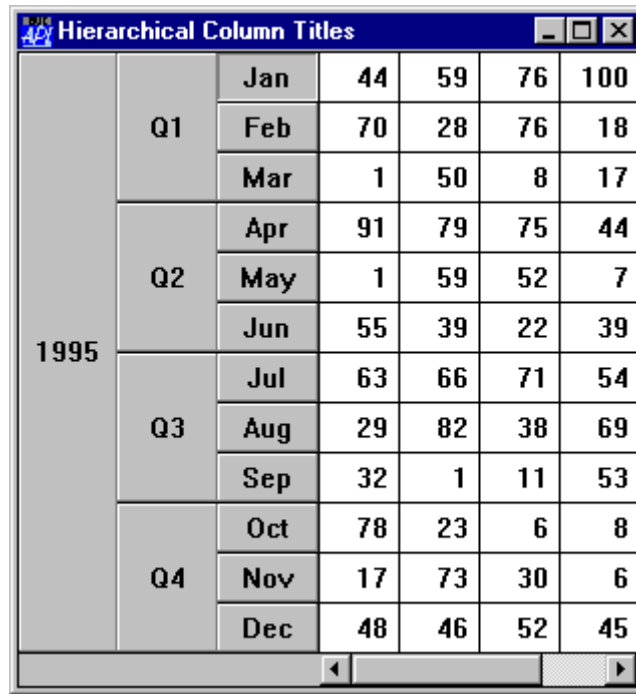
It is an integer vector with the same length as the RowTitles property. A value of 0 indicates that the corresponding element of RowTitles is a top-level title. A value of 1 indicates that the corresponding title is a sub-title of the most recent title whose RowTitleDepth is 0; a value of 2 indicates that the corresponding title is a sub-title of the most recent title whose RowTitleDepth is 1, and so forth. For example:

```
'F'□WC'Form'('Coord' 'Pixel')('Size' 318 310)
'F'□WS'Caption' 'Hierarchical Column Titles'
'F.G'□WC'Grid'(?12 4p100)(0 0)(318 310)
'F.G'□WS('TitleWidth' 150)('TitleHeight' 0)
'F.G'□WS'CellWidths' 40

Q1←'Q1' 'Jan' 'Feb' 'Mar'
Q2←'Q2' 'Apr' 'May' 'Jun'
Q3←'Q3' 'Jul' 'Aug' 'Sep'
Q4←'Q4' 'Oct' 'Nov' 'Dec'
RT←(c'1995'),Q1,Q2,Q3,Q4
RD←0,16p1 2 2 2

'F.G'□WS('RowTitles'RT)('RowTitleDepth'RD)

'F.G'□WS'RowTitleAlign' 'Centre'
```



1995	Q1	Jan	44	59	76	100
		Feb	70	28	76	18
		Mar	1	50	8	17
	Q2	Apr	91	79	75	44
		May	1	59	52	7
		Jun	55	39	22	39
	Q3	Jul	63	66	71	54
		Aug	29	82	38	69
		Sep	32	1	11	53
	Q4	Oct	78	23	6	8
		Nov	17	73	30	6
		Dec	48	46	52	45

Note that the [Section 6.358](#) method is not supported in combination with hierarchical row titles.

6.499 RowTitleFCol Property

Applies To: [Section 6.263](#)

Description

The RowTitleFCol property specifies the colour of the row titles in a [Section 6.263](#) object

RowTitleFCol may be a scalar that specifies a single colour to be used for all of the row titles, or a vector that specifies the colour of each of the row titles individually. An element of RowTitleFCol may be an enclosed 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or it may be a scalar that defines a standard Windows colour element (see [Section 6.45](#) for details). Its default value is 0 which obtains the colour defined for Button text.

6.500 RowTitles Property

Applies To: [Section 6.263](#)

Description

This property specifies the headings that are displayed to the left of the rows in a [Section 6.263](#) object. If specified, it must be a vector of character vectors or matrices whose length is the same as the number of rows implied by the [Section 6.642](#) property. The default value of RowTitles is an empty character vector. In this case, the system displays the row numbers.

To disable the display of row titles in a [Section 6.263](#), you should set the [Section 6.619](#) property to 0.

6.501 RowTreeDepth Property

Applies To: [Section 6.263](#)

Description

The RowTreeDepth property specifies the structure of the rows in a [Section 6.263](#) object. It is either a scalar 0 or an integer vector of the same length as the number of rows in the [Section 6.263](#). RowTreeDepth is similar to the [Section 6.156](#) property of the [Section 6.629](#) object.

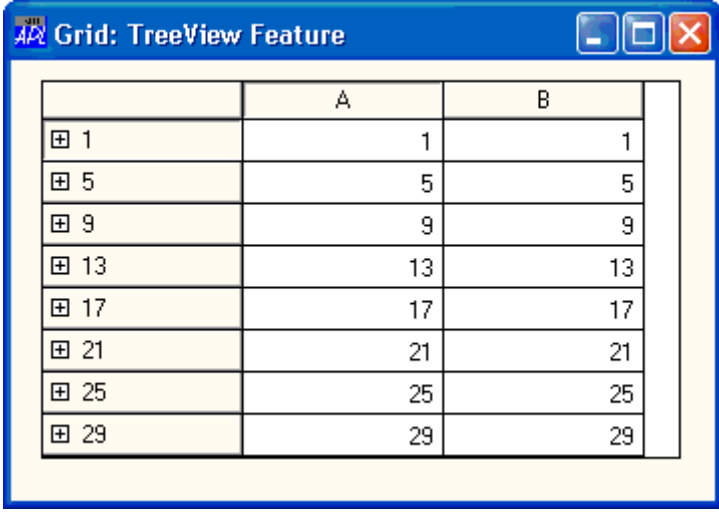
A value of 0 indicates that the corresponding row is a top-level row. A value of 1 indicates that the corresponding row is a child of the most recent row whose RowTreeDepth is 0; a value of 2 indicates that the corresponding row is a child of the most recent row whose RowTreeDepth is 1, and so forth.

When you set RowTreeDepth, the [Section 6.263](#) is redrawn so that only rows with a RowTreeDepth of 0 are visible.

The [Section 6.495](#) method can be used to make data visible to a specific depth.

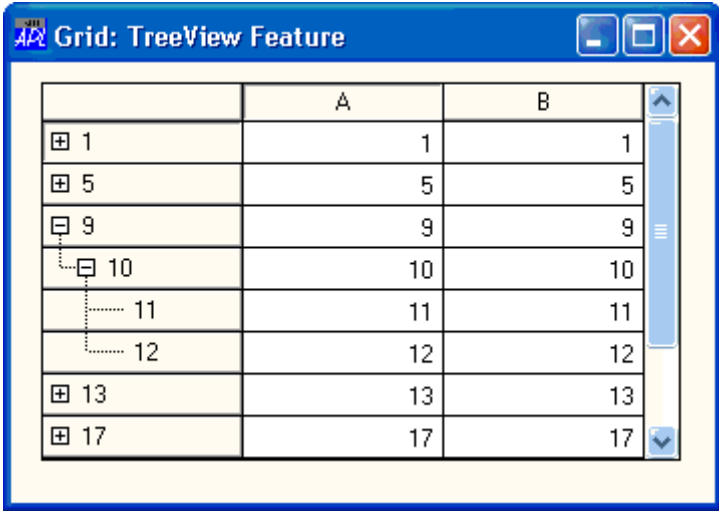
EXAMPLE

```
'F'WC'Form' 'Grid: TreeView Feature'
'F.G'WC'Grid'(30 2p2/130)
F.G.RowTreeDepth←30p0 1 2 2
```



	A	B
+ 1	1	1
+ 5	5	5
+ 9	9	9
+ 13	13	13
+ 17	17	17
+ 21	21	21
+ 25	25	25
+ 29	29	29

The user can interact with the tree images to expand and contract rows of the [Section 6.263](#).



	A	B
+ 1	1	1
+ 5	5	5
- 9	9	9
- 10	10	10
- 11	11	11
- 12	12	12
+ 13	13	13
+ 17	17	17

6.502 RowTreeImages Property

Applies To: [Section 6.263](#)

Description

The RowTreeImages property is a simple character vector or ref, or a vector of character vectors or refs, that specifies the names(s) of, or ref(s) to, Bitmap objects that are used to display the tree nodes for a [Section 6.263](#) object.

Note that images in tree nodes are only displayed if [Section 6.503](#) is set to 'ImagesOnly', 'ImagesAndLines', or 'AllImagesAndLines'.

If RowTreeImages is not specified default images are used.

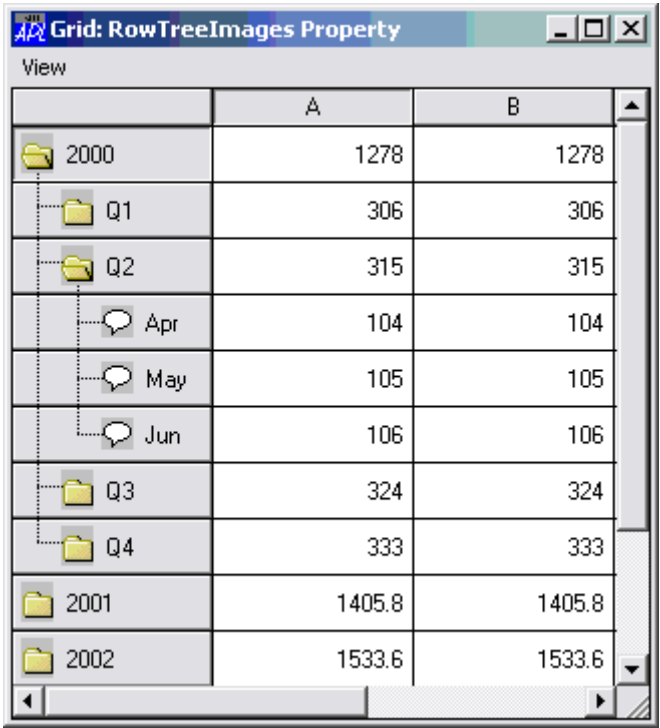
The Bitmap specified by the 1st element of RowTreeImages is used to display *unopened* nodes.

The Bitmap specified by the 2nd element of RowTreeImages is used to display *opened* nodes.

The Bitmap specified by the 3rd element of RowTreeImages is used to display nodes *without children*.

EXAMPLE

```
'Closed'␣WC'Bitmap' 'Folder.bmp'
'Open'␣WC'Bitmap' 'FolderOpen.bmp'
'Item'␣WC'Bitmap' 'Ideas'
F.G.RowTreeStyle←'AllImagesAndLines'
F.G.RowTreeImages←'Closed' 'Open' 'Item'
```



The screenshot shows a window titled "Grid: RowTreeImages Property". Inside, there is a "View" section displaying a hierarchical tree structure. The tree has a root node "2000" which branches into "Q1", "Q2", "Q3", and "Q4". "Q1" further branches into "Apr", "May", and "Jun". Each node is represented by a folder icon. To the right of the tree is a table with two columns, "A" and "B". The values in column "A" are 1278, 306, 315, 104, 105, 106, 324, 333, 1405.8, and 1533.6. The values in column "B" are 1278, 306, 315, 104, 105, 106, 324, 333, 1405.8, and 1533.6. The table has a scrollbar on the right side.

	A	B
2000	1278	1278
Q1	306	306
Q2	315	315
Apr	104	104
May	105	105
Jun	106	106
Q3	324	324
Q4	333	333
2001	1405.8	1405.8
2002	1533.6	1533.6

6.503 RowTreeStyle Property

Applies To: [Section 6.263](#)

Description

RowTreeStyle specifies the visible attributes of the tree displayed in the Row titles of a [Section 6.263](#).

The value of the RowTreeStyle property is a character vector chosen from the following :

'LinesOnly '	Only the lines of the tree structure are drawn.
'ImagesOnly '	Only the images of nodes with children are drawn.
'ImagesAndLines '	Both lines and images for nodes with children are drawn.
'AllImagesOnly '	Images for all nodes are drawn.
'AllImagesAndLines '	Both lines and images for all nodes are drawn.

The default value, 'ImagesAndLines', is illustrated in the first picture below. Other values are displayed in subsequent pictures.

The screenshot shows a software window titled "RowTreeStyle: ImagesAndLines". It displays a hierarchical tree view on the left and a corresponding data table on the right. The tree view includes expand/collapse icons and small images next to the node names. The table has three columns: a blank header, "A", and "B".

	A	B
2000	1278	1278
Q1	306	306
Jan	101	101
Feb	102	102
Mar	103	103
Q2	315	315
Apr	104	104
May	105	105
Jun	106	106
Q3	324	324

`F.G.RowTreeStyle←'LinesOnly'`

The screenshot shows a software window titled "RowTreeStyle: LinesOnly". It displays a hierarchical tree view on the left and a corresponding data table on the right. The tree view includes expand/collapse icons but lacks the small images seen in the previous screenshot. The table structure is identical to the first screenshot.

	A	B
2000	1278	1278
Q1	306	306
Jan	101	101
Feb	102	102
Mar	103	103
Q2	315	315
Apr	104	104
May	105	105
Jun	106	106
Q3	324	324

`f.g.RowTreeStyle←'ImagesOnly'`

RowTreeStyle: ImagesOnly

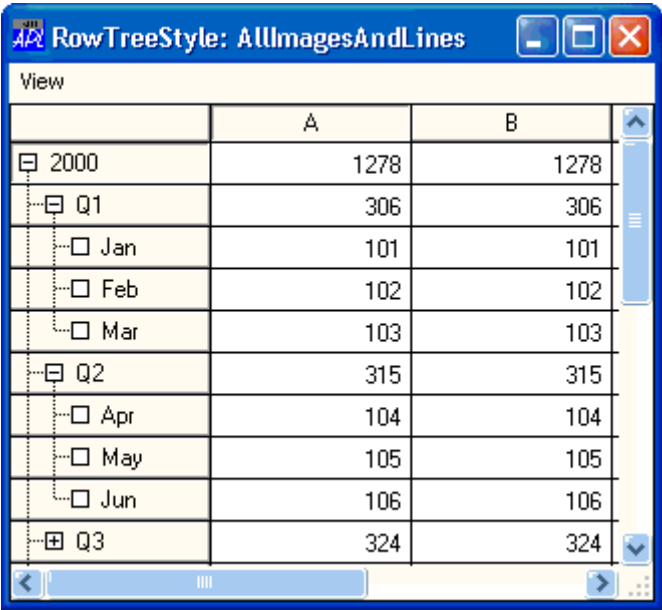
View			
	A	B	
[-] 2000	1278	1278	
[-] Q1	306	306	
Jan	101	101	
Feb	102	102	
Mar	103	103	
[-] Q2	315	315	
Apr	104	104	
May	105	105	
Jun	106	106	
[+] Q3	324	324	

```
f.g.RowTreeStyle←'AllImagesOnly'
```

RowTreeStyle: AllImagesOnly

View			
	A	B	
[-] 2000	1278	1278	
[-] Q1	306	306	
[-] Jan	101	101	
[-] Feb	102	102	
[-] Mar	103	103	
[-] Q2	315	315	
[-] Apr	104	104	
[-] May	105	105	
[-] Jun	106	106	
[+] Q3	324	324	

```
f.g.RowTreeStyle←'AllImagesAndLines'
```



6.504 RTFPrint Method 461

Applies To: [Section 6.488](#)

Description

This method is used to print the contents ([Section 6.506](#)) of a [Section 6.488](#) object.

The argument to RTFPrint is $\emptyset$, or a 1 to 4-element array as follows:

[1]	Printer name	Optional - character vector (see below)
[2]	Print range	Optional - (see below)
[3]	Number of copies	Optional - Integer.
[4]	Collate	Optional - 0 or 1

Printer name may be the name of an existing Printer object, or the (Windows) name of an installed printer. If you use the latter, the document will be spooled immediately. An empty vector implies the default printer.

Print range may be a simple character vector containing 'All', 'Pages', or 'Selection'. Alternatively, it may be a 3 or 4-element nested vector containing:

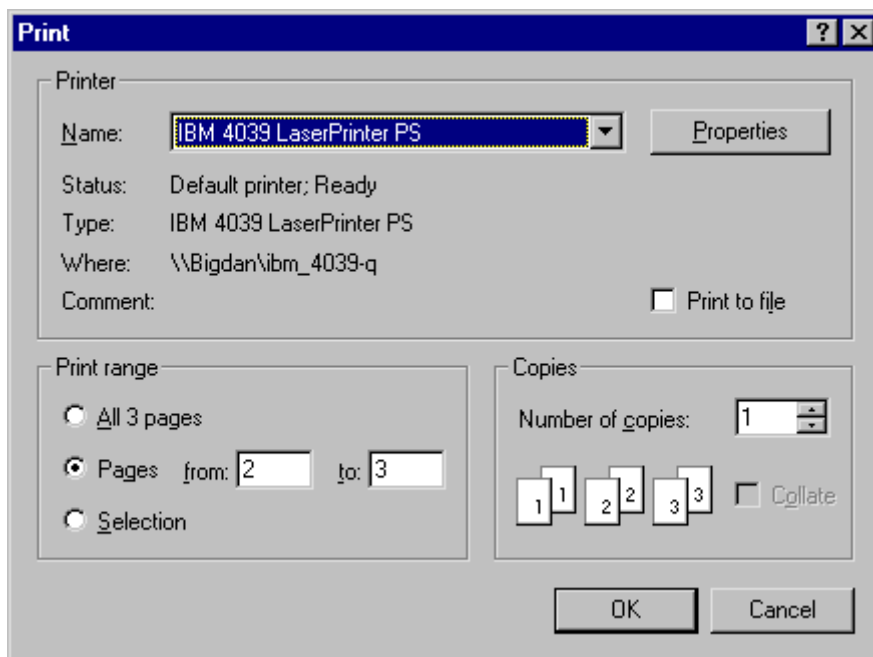
[1]	'All', 'Pages', Or 'Selection'
[2]	Start page (integer)
[3]	End page (integer)
[4]	Maximum pages (ignored)

6.505 RTFPrintSetup Method 460

Applies To: [Section 6.455](#), [Section 6.488](#)

Description

This method is used to display a print set-up dialog box. The dialog box allows the user to select a particular printer, the pages to be printed and other information. The user's choices are returned in the result.



The argument to RTFPrintSetup is θ , or a 1 to 3-element array as follows:

[1]	Print range	Optional - (see below)
[2]	Number of copies	Optional - Integer.
[3]	Collate	Optional - 0 or 1

Print range may be a simple character vector containing 'All', 'Pages', or 'Selection'.

Alternatively, it may be a 3 or 4-element nested vector containing:

[1]	'All', 'Pages', or 'Selection'
[2]	Start page (integer)
[3]	End page (integer)
[4]	Maximum pages

Maximum pages (4th element of Print range) may be an integer number, or the name of a reference object. The latter allows the system to calculate the total number of pages required. If the object to which the RTFPrintSetup event is sent is a [Section 6.488](#), this is the name of a printer object. If the object to which the RTFPrintSetup event is sent is a Printer, this is the name of a [Section 6.488](#) object. Both are required because the number of pages of a printed document is dependent upon both the content of the document and the characteristics of the device upon which it will be printed.

If the user presses OK, the result is a 4-element vector containing the user's choices as follows:

[1]	Printer name	character vector
[2]	Print range	(see above)
[3]	Number of copies	Integer
[4]	Collate	0 or 1

EXAMPLE

```
F.T.RTFPrintSetup ('ALL' 1 1 'PR')
IBM 4039 LaserPrinter PS    Pages 2 3 3 1 0
```

6.506 RTFText Property

Applies To: [Section 6.99](#), [Section 6.488](#)

Description

The RTFText property is used to set or retrieve the contents of a [Section 6.99](#) or a [Section 6.488](#) object in rich text format (RTF). It is always a character vector.

6.507 RunMode Property

Applies To: [Section 6.420](#)

Description

This property specifies the way in which an [Section 6.420](#) object serves multiple clients.

RunMode is a character vector and may be 'MultiUse' (the default), 'SingleUse' or 'RunningObject'.

If RunMode is 'MultiUse', OLE will load a single copy of Dyalog APL and the appropriate workspace into memory. All OLE client processes will communicate with the same Dyalog APL session.

Note that in this case, each OLE client is actually connected to a separate *instance* of the corresponding APL namespace. That is to say, each client will appear to have its own private copy of the namespace. However, the individual functions and variables in the namespace are not physically copied until they are changed. This means that, in general, OLE clients will share APL functions but have private copies of the namespace variables. However, please remember that global objects in the workspace or in other namespaces are not *instanced* and will effectively be shared by all clients although they are not directly accessible to them.

If RunMode is 'SingleUse', OLE will load a separate copy of Dyalog and a separate copy of the appropriate workspace into memory for each OLE client. Each OLE client operates directly on the namespace associated with the object and not an *instance* of it.

If RunMode is 'RunningObject', OLE will load a single copy of Dyalog and the appropriate workspace into memory. All OLE client processes will communicate with the same Dyalog session and indeed with the same namespace. The namespace is not *instanced* and all objects, including exported variables, are shared by all clients.

6.508 Scroll Object

Purpose: Provides a vertical or horizontal scrollbar.

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm ToolBar
ToolControl

Children Circle Cursor Ellipse Font Marker Poly Rect Text Timer

Properties Type Posn Size Coord Align Border Active Visible Event Thumb Range Step
VScroll HScroll Limits Sizeable Draggable BCol CursorObj AutoConf Data Attach EdgeStyle
Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles KeepOnClose Redraw
TabIndex PageSize MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose
Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick
MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate
GestureTwoFingerTap GesturePressAndTap Select ThumbDrag Scroll

Description

The Scroll object provides a vertical or horizontal scrollbar that can be used as a "free-standing" object or can be "attached" to the side of its parent.

An "attached" scrollbar is one that extends along one edge of a [Section 6.227](#), [Section 6.578](#) or [Section 6.279](#) and has a standard width or height. When the [Section 6.227](#) or [Section 6.279](#) is resized, a vertical attached scrollbar is resized vertically but remains the same width and stays fixed to the side of its parent. Similarly, a horizontal attached scrollbar is resized horizontally but remains the same height.

For most purposes, the use of the Scroll object to provide attached scrollbars in a [Section 6.227](#) has been superseded by the provision of scrollbars as a **property** of a [Section 6.227](#).

A "free-standing" scrollbar is typically used as a "scale" for selecting a numeric value from a range and may appear and behave rather differently from a standard attached scrollbar. Firstly, a free-standing scrollbar will normally be positioned at an arbitrary position within its parent [Section 6.227](#) or [Section 6.279](#) and be associated with other objects such as [Section 6.341s](#) and [Section 6.185](#) fields. Secondly, when its parent [Section 6.227](#) or [Section 6.279](#) is resized, it is probably desirable that the scrollbar reacts in the same way as the other child objects, so that the overall appearance of the layout is maintained.

The [Section 6.13](#) property determines whether or not a scrollbar is attached, and if so, to which side of the parent [Section 6.279](#) or [Section 6.227](#) it is fixed. The direction of the scrollbar is determined by the [Section 6.646](#) and [Section 6.303](#) properties, which

are mutually exclusive. The position and size of the scrollbar are determined by [Section 6.452](#) and [Section 6.556](#).

To obtain an "attached" scrollbar, it is sufficient for most purposes to specify only the [Section 6.13](#) property. If so, the direction of the scrollbar and its position and size (which are otherwise defined by [Section 6.646](#), [Section 6.303](#), [Section 6.452](#) and [Section 6.556](#)) are determined automatically for you.

To obtain a "free-standing" scrollbar, it is recommended for most purposes that you set [Section 6.13](#) to 'None' and define the orientation, position and size of the scrollbar explicitly using [Section 6.646](#) or [Section 6.303](#), [Section 6.452](#) and [Section 6.556](#).

[Section 6.646](#) and [Section 6.303](#) may only be set when the object is created and may not subsequently be changed.

If you do attach a "free-standing" scrollbar to a particular side of its parent using [Section 6.13](#), it will maintain its physical position (in pixels) relative to the side to which it is attached, and its dimension in that direction will remain fixed.

The [Section 6.13](#) property is a character vector containing 'Top', 'Bottom', 'Right', 'Left' or 'None'. If you specify [Section 6.13](#) 'Right' you get a vertical scrollbar attached to the right-hand edge of the parent [Section 6.227](#) or [Section 6.279](#). [Section 6.13](#) 'Left' also produces a vertical scrollbar, but one that is attached to the left-hand edge. [Section 6.13](#) 'Top' and 'Bottom' each produce horizontal scrollbars, attached to the top and bottom edges of the [Section 6.227](#) or [Section 6.279](#).

Note that the default value of [Section 6.13](#) is 'Right' unless [Section 6.303](#) is set to $\bar{1}$ in which case it is 'Bottom'. It must therefore be explicitly set to 'None' if you want a non-attached "free-standing" scrollbar.

[Section 6.646](#) and [Section 6.303](#) are used to specify the orientation of the scrollbar explicitly, usually in conjunction with [Section 6.13](#) set to 'None'. [Section 6.646](#) or [Section 6.303](#) may be specified when the object is created by *Dyalog APL Language: Wc*, but cannot be changed using *Dyalog APL Language: Ws*. The two properties are mutually exclusive. Each of them may be set to 0 or $\bar{1}$, where $\bar{1}$ means "true" and 0 means "false". Thus ([Section 6.646](#) $\bar{1}$) defines a vertical scrollbar, while ([Section 6.303](#) $\bar{1}$) specifies a horizontal one. Setting either property to $\bar{1}$ automatically causes the other to be set to 0. If you try to set both to $\bar{1}$, [Section 6.646](#) takes precedence and [Section 6.303](#) is reset to 0.

[Note: the reason for using two properties where one would be sufficient is to allow for the possible future implementation of scrolling groups as provided by $\square\text{SM}/\square\text{SR}$.]

Scrolling is controlled by the [Section 6.608](#), [Section 6.469](#) and [Section 6.576](#) properties.

[Section 6.608](#) sets and reports the current position of the "thumb" as an integer in the range 1 to the value of the [Section 6.469](#) property.

[Section 6.576](#) determines the size of changes reported when the user clicks a scroll arrow (small change) or clicks on the body of the scrollbar (large change). [Section 6.576](#) is a 2-element numeric vector whose first element specifies the value of the "small change" and whose second element specifies the value of the "large change". The [Section 6.438](#) property specifies the sizes of the thumb in the scrollbar.

Examples of Attached Scrollbars

```
'F' □WC 'Form' 'Default Scroll Bar'
'F.SCR' □WC 'Scroll'

'F' □WC 'Form' 'Default Horizontal Scroll Bar'
'F.SCR' □WC 'Scroll' ('HScroll' -1)
```

Examples of Free-Standing Scrollbars

```
'F' □WC 'Form' 'Non-Default Scroll Bar'
'F.SCR' □WC 'Scroll' (5 45)(90 10)
('Align' 'None')('VScroll' -1)

'F' □WC 'Form' 'Horizontal Scroll Bars'
'F.SC1' □WC 'Scroll' (15 15)(15 70)
('Align' 'None')('HScroll' -1)
'F.SC2' □WC 'Scroll' (40 40)(20 20)
('Align' 'None')('HScroll' -1)
'F.SC3' □WC 'Scroll' (85 5)(10 90)
('Align' 'None')('HScroll' -1)
```

6.509 Scroll Event 37

Applies To: [Section 6.508](#), [Section 6.625](#)

Description

If enabled, this event is generated when the user attempts to move the thumb in a scrollbar. This can be done in one of three ways:

1. dragging the thumb.
2. clicking in one of the "arrow" buttons situated at the ends of the scrollbar. This is termed a small change, the size of which is defined by [Section 6.576](#)[1].

3. clicking in the body of the scrollbar. This is termed a large change, the size of which is defined by [Section 6.576](#)[2].

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Scroll' or 37
[3]	Scroll Type	numeric
[4]	Position	numeric

The value of Scroll Type is 0 (drag), 1 or -1 (small change) or 2 or -2 (large change). The sign indicates the direction.

The value of Position is the new (requested) position of the thumb. Notice however, that the event is generated **before** the thumb is actually moved. If your callback function returns a scalar 0, the position of the thumb will remain unaltered.

6.510 ScrollOpposite Property

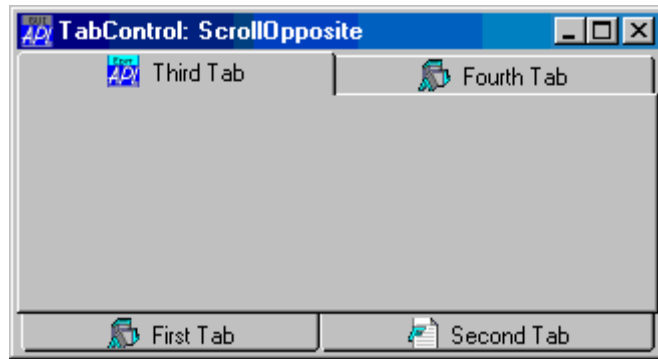
Applies To: [Section 6.585](#)

Description

The ScrollOpposite property specifies that unneeded tabs scroll to the opposite side of a [Section 6.585](#), when a tab is selected.

ScrollOpposite is a single number with the value 0 (normal scrolling) or 1 (scrolling to the opposite side); the default is 0.

The picture below illustrates a [Section 6.585](#) with ScrollOpposite set to 1, after the user has clicked *Third Tab*.



Setting ScrollOpposite to 1 implies that [Section 6.404](#) is also 1. If you set ScrollOpposite to 1 in a `WC` statement, the [Section 6.404](#) property will automatically be set to 1, even if you try to set [Section 6.404](#) to 0 in the same statement. If you subsequently change [Section 6.404](#) back to 0 using `WS`, this will work, but the effect is not useful and it is not supported.

6.511 SelDate Property

Applies To: [Section 6.58](#)

Description

The SelDate property identifies the range of dates that is currently selected in a [Section 6.58](#) object.

SelDate is a 2-element integer vector of [Section 8.5](#) values that identifies the first and last dates that are currently selected.

6.512 SelDateChange Event 265

Applies To: [Section 6.58](#)

Description

If enabled, this event is reported when the user changes the date, or range of dates, that is selected in a [Section 6.58](#) object. This event is also reported when the [Section 6.58](#) object is scrolled and the selection changes automatically to another month.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to -1 or by returning 0 from a callback function.

The event message reported as the result of `DQ`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'SelDateChange' or 265
[3]	First Date	an integer (IDN)
[4]	Last Date	an integer (IDN)

6.513 Select Event 30

Applies To

Description

For a [Section 6.55](#) with [Section 6.577](#) 'Push' this event is generated when the user "pushes" the button. This can be done by clicking the left mouse button, or by pressing the Enter key or the space bar when the [Section 6.55](#) has the focus. The Select event can also be generated when the [Section 6.55](#) does not have the focus, by pressing the Enter key when its [Section 6.148](#) property is 1 or by pressing the ESC key when its [Section 6.64](#) property is 1.

For a [Section 6.55](#) with [Section 6.577](#) 'Radio' or 'Check' this event is generated when the user toggles the button from one state to another. This can be achieved by clicking the left mouse button or by pressing the space bar when the [Section 6.55](#) has the focus.

For a [Section 6.121](#) or [Section 6.346](#) object, a Select event is generated when the user selects an item from the list, whether by pressing the arrow keys or by clicking the left mouse button.

For a [Section 6.384](#), a Select event is generated when the user chooses the item.

For all other objects, this event is generated when the user presses the keys associated with the object's [Section 6.2](#) property.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	Event code

6.514 SelectCertificate Event 848

Applies To: [Section 6.307](#)

Description

This event is triggered when [Section 6.307](#) requests a resource from a server that requires a certificate.

The event message reported as the result of `□□□□□□□□`, or supplied as the right argument to your callback function, is a 7-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'SelectCertificate' or 848
[3]	Index	Integer (see below)
[4]	Addr	Host address
[5]	Port	Host port
[6]		'is proxy'
[7]	Certificates	See below

Certificates is a vector of namespaces, each of which represents an available certificate and contains the following variables:

Name	Description
DER	Distinguished Encoding Rules. Character Vector.
Subject	Namespace (see below)
Issuer	Namespace (see below)
SerialNumber	Integer

The **Subject** and **Issuer** namespaces contain the following variables:

Name	Description
CommonName	Character vector
CountryName	Character vector
DisplayName	Character vector

The application should respond to this event by selecting a certificate from the list of available certificates reported by the 7th element of the event message. This is done by having a callback function that sets the 3rd element of the event message (Index) to the 0-origin index in Certificates and returns the event message as its result.

EXAMPLE

```

    ▾ arg←cb arg
[1]   a SelectCertificate callback function
[2]   arg[3]←0 a Select the first certificate
    ▾

```

6.515 SelectionBorderWidth Property

Applies To: [Section 6.263](#)

Description

The SelectionBorderWidth property specifies the width of the border that is drawn around the currently selected block of cells. It is expressed in pixels.

6.516 SelectionColor Property

Applies To: [Section 6.263](#)

Description

The SelectionColor property specifies the colour used to highlight the currently selected block of cells and, if [Section 6.298](#) is 1, the corresponding row and column headings. See also [Section 6.517](#).

6.517 SelectionColorAlpha Property

Applies To: [Section 6.263](#)

Description

The SelectionColorAlpha property is a 2-element integer vector that specifies the degree of transparency or shade of the colour that is used to highlight the currently selected block of cells in a [Section 6.263](#). See [Section 6.516](#).

The first element refers to the shade to be used when the Grid has the input focus; the second to when it doesn't. Each element is an integer in the range 0 (invisibly light) to 255 (fully dark).

6.518 SellmageIndex Property

Applies To: [Section 6.122](#), [Section 6.629](#)

Description

The SellmageIndex property determines which bitmapped images in an [Section 6.316](#) correspond to items in a [Section 6.629](#) object when the item is *selected*. It is an integer vector whose length is the same as the number of items in the object and is $\square$ IO dependent.

See also [Section 6.315](#)

6.519 SellItems Property

Applies To: [Section 6.121](#), [Section 6.122](#), [Section 6.263](#), [Section 6.346](#), [Section 6.348](#), [Section 6.629](#)

Description

This property determines which (if any) of the items in an object are currently selected and highlighted. Except in a Grid, it is a Boolean vector with one element per item in the list. A value of 1 means "selected"; 0 means "not selected".

This property is used after a [Section 6.513](#) event to identify which item has been chosen. In a [Section 6.121](#) or a [Section 6.346](#) with [Section 6.577](#) 'Single' it will contain only a single 1.

SellItems should also be used to pre-set the contents of the edit field in a [Section 6.121](#) box with [Section 6.577](#) 'Drop'. In [Section 6.121](#) boxes with [Section 6.577](#) 'Simple' or 'DropEdit', the contents of the edit field may also be specified by the [Section 6.606](#) property. If you specify both, the value of [Section 6.606](#) takes precedence.

In a [Section 6.263](#) SellItems is a 2-element integer vector. The first element identifies the row and column coordinates respectively of the first cell(s) in the selected block(s) and the second, the row and column coordinates of the last cell(s) in the selected block(s). If the [Section 6.80](#) property allows only a single block of cells to be selected, each set of coordinates is a 2-element vector. If the [Section 6.80](#) property permits more

than one block of cells to be selected, each set of coordinates is a 2-column matrix with one row per selected block, whose columns identify the first row and column, and the last row and column respectively of each selected block.

6.520 SelRange Property

Applies To: [Section 6.625](#)

Description

The SelRange property specifies the selected range in a TrackBar which has Style 'Selection'. It is a 2-element numeric vector.

6.521 SelText Property

Applies To: [Section 6.56](#), [Section 6.121](#), [Section 6.122](#), [Section 6.185](#), [Section 6.488](#)

Description

This property determines or identifies the portion of text in an object that is currently selected and highlighted. It can be used to pre-select all or part of the text to be replaced or deleted when the user starts typing. It can also be used to query the area of text that the user has highlighted. This can be useful if you want to implement your own cut/paste/replace features.

SelText is always a 2-element integer vector. If the field contents (defined by the [Section 6.606](#) property) is a vector, SelText is simple. Its first element is the index of the first selected character and its second element is 1 + the index of the last selected character. The length of the selected string is therefore obtained by subtracting the first element from the second.

If there are no characters selected, the two elements are equal and specify the current position of the input cursor.

If the contents is a vector of vectors or a matrix, each element of SelText is a 2-element vector. The first item in each of the elements indexes the vector (in a vector of vectors) or row (in a matrix). The second item in each element indexes the position of the character in the vector or along the row. Again, the value reported for the last character in the selected string is 1 + its index.

6.522 Separator Object

Purpose: A horizontal or vertical line used to separate items in a menu.

Parents Menu MenuBar

Children Timer

Properties Type Posn Style Event FCol BCol Data EdgeStyle Translate KeepOnClose
MethodList ChildList EventList PropList

Methods Detach

Events Close Create

Description

This object provides a vertical or horizontal line to separate items in a [Section 6.382](#). It may also be used to split a [Section 6.383](#) over more than one line.

The orientation of the Separator is determined by its [Section 6.577](#) property, which may be 'Horz' (horizontal) or 'Vert' (vertical). The default is 'Horz'.

If you want to provide a menu with a 3-Dimensional (pushbutton) appearance, you should also set the [Section 6.184](#) property on any Separator objects in it. Alternatively, you can achieve the same effect by setting the background colour ([Section 6.45](#)) for the Separators to grey (192 192 192).

The [Section 6.452](#) property is a single integer number which specifies the positional index of the Separator relative to the other objects in the [Section 6.382](#). A Separator does not generate any events.

Like other components of a menu, the position of a Separator is normally determined by the order in which it is created in relation to other objects with the same parent. However, you can use the [Section 6.452](#) property to **insert** a Separator into an existing structure. For example, having defined three [Section 6.384](#) objects as children of a [Section 6.382](#), you can insert a Separator between the first and the second by specifying its [Section 6.452](#) to be 2. Note that the value of [Section 6.452](#) for the [Section 6.384](#)s that were previously second and third will then be reset to third and fourth respectively.

If you put a Separator (either [Section 6.577](#)) into a [Section 6.383](#), it has the effect of adding another line to it. Any items added after the Separator will appear in the new line.

6.523 ServerVersion Property

Applies To: [Section 6.420](#)

Description

This property specifies the version number of an [Section 6.420](#) object.

It is a 2-element integer vector that specifies the major and minor version numbers respectively.

The default value of ServerVersion is (1 0).

6.524 ServiceNotification Event 94

Applies To: [Section 6.489](#)

Description

This event is generated in an APL service whenever the Windows Service Control Manager (SCM) requests a change of state. See *Installation/Configuration: Apl Application As A Service*.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ServiceNotification' or 94
[3]	Action	Integer.
[4]	State	7-element integer vector

For further details, see the on-line documentation for SERVICE_STATE and the function HashDefine in the sample workspace `apl service`.

The state of a Windows service is determined by the user and the Windows Service Control Manager (SCM).

When the SCM requests a change of state, the APL interpreter responds by setting its state to the corresponding *pending* level (SERVICE_STOPPED_PENDING, SERVICE_RUNNING_PENDING or SERVICE_PAUSED_PENDING) and then generates a [Section 6.524](#) event.

It is the responsibility of the APL service to process this event, perform the appropriate application tasks, and then respond (to the SCM) by calling the [Section 6.537](#) method to confirm that the service has reached the desired state.

6.525 SetCellSet Method 171

Applies To: [Section 6.263](#)

Description

The SetCellSet method sets the value of the [Section 6.81](#) property of a [Section 6.263](#) for a particular cell.

The argument to SetCellSet is a 3-element array as follows:

[1]	Row	integer
[2]	Column	integer
[3]	Value	0 or 1

6.526 SetCellType Method 156

Applies To: [Section 6.263](#)

Description

This method is used to change the type of a particular cell in a [Section 6.263](#).

The argument to SetCellType is a 3-element vector as follows :

[1]	Cell row	integer
[2]	Cell column	integer
[3]	Cell type	integer

6.527 SetColSize Event 176

Applies To: [Section 6.263](#), [Section 6.348](#)

Description

If enabled, this event is reported when the user changes the width of a column in a [Section 6.263](#) or [Section 6.348](#) object, or changes the width of the row titles in a [Section 6.263](#). This may be done by dragging a border with the mouse or by double-clicking over a border. In the former case, the default action is to adjust the width of the appropriate column or the width of the row title area to the size selected by the user. In the latter case, the default action is to adjust the width to the maximum required to display all the data.

In either case, you can disable the default action by setting the event action code to `-1` or you can selectively prevent a particular resize operation from taking place by returning 0 from a callback function.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'SetColSize' or 176
[3]	Column number	Integer. This is sensitive to the index origin, <code>⌈IO</code> , but is <code>-1</code> if the user has resized the row titles in a Grid.
[4]	Width	Integer containing the value of the (new)column width. This is <code>-3</code> if the user has double-clicked to request automatic width adjustment.
[5]	Undo flag	0 or 1

You can resize a column or resize the row titles under program control by calling `SetColSize` as a method. If you specify `-1` as the *Width* parameter, the column will be resized to its default width .. If you specify a value of `-2` the column will be resized to fit the data. The following expression will size the first `NCOLS` columns of a Grid called `F.G` to fit the data and the column titles:

```
{F.G.SetColSize ⍵ -3}¨1:NCOLS
```

The Undo flag is applicable only to a Grid object and is always 1 if the event was generated by the user.

6.528 SetEventInfo Method 547

Applies To: [Section 6.7](#), [Section 6.420](#)

Description

This method is used to register an event that may be generated by an [Section 6.7](#) or [Section 6.420](#) object.

A host application that wishes to attach a callback function to an event in a Dyalog APL [Section 6.7](#) or [Section 6.420](#), needs to know the name of the event and the number and data types of any parameters that the event may supply. It also needs to know the data type (if any) of the result that the callback function may be expected to pass back to the control.

An [Section 6.7](#) or [Section 6.420](#) generates an event in the host application using `⋄ ⌈NQ`. The right argument is a vector whose first 2 elements are character vectors containing the names of the [Section 6.7](#) or [Section 6.420](#) and the event respectively. The parameters for the event are passed as additional elements in the argument.

Another way to think about it is that when you generate an event using `⋄ ⌈NQ`, you are effectively calling a function, of your specification, in the host application. To enable the host application to accept the function call, it needs to know the number of parameters that you will supply and their data types.

A further consideration is that if you specify that the data type of a parameter is a *pointer* (for example, `'VT_PTR TO I4'`) this will allow a callback function to modify the parameter in-situ. If so, the result returned by `⋄ ⌈NQ` will be the modified values of any such parameters; this is a similar mechanism to `⌈NA`.

The argument to `SetEventInfo` is a 1, 2, 3 or 4-element array as follows:

[1]	Event name	character vector
[2]	Event info	nested array (see below)
[3]	Help ID	integer
[4]	DISPID	integer. See DISPID (Dispatch ID)

Event info

Event info, specifies an optional help string which describes what the event does, the data type of the result (if any) and the names and data types of its arguments.

If the event is fully described, each element of Event Info is a 2-element vector of character vectors. The first element contains the help string and the [Section 8.2](#) of the result that the callback function in the host application is expected to supply. Subsequent elements contain the name and [Section 8.2](#) of each of the parameters supplied by the event.

However, both the help string and the names of the parameters are optional and may be omitted. If so, one or more elements of *Event Info* may be a simple character vector.

Help ID

This is an integer value that identifies the help context id for the event within the help file associated with the *HelpFile* property of the [Section 6.7](#) object. The value -1 means that no help is provided. APL stores this information in the registry from where it may be retrieved by the host application.

EXAMPLE

The example *Dual* ActiveXControl, that is fully described elsewhere, generates a *ChangeValue1* event. This event occurs when the user moves the thumb in a *TrackBar* that is internal to an instance of the [Section 6.7](#).

The external *ChangeValue1* event is fired by an internal APL callback function (called *ChangeValue*) that is attached to *ThumbDrag* and *Scroll* events on the *TrackBar* object. The internal callback function is :

```
[0]   ChangeValue MSG
[1]   A Callback for ThumbDrag and Scroll
[2]   Value1←4 □NQ ' ' 'ChangeValue1' (⌵-1↑MSG)
[3]   CalcValue2
[4]   'V1'□WS'Text' (⌵Value1)
[5]   'V2'□WS'Text' (⌵Value2)
```

Note that *ChangeValue*[2] generates the external *ChangeValue1* event by invoking 4 □NQ, passing it the new value provided by the *TrackBar*. However, the host application is permitted to modify that value, returning it in the result of 4 □NQ. This result, rather than the *TrackBar* value itself, is then used to update other (Label) controls in the object.

The following statements were used to declare the *ChangeValue1* event. The event provides a single parameter named *Value1* that may be modified in-situ by a callback function in the host application. The callback is not, otherwise, expected to return a result.

```
INFO←'Occurs when the value of the control is changed' 'VT_VOID'
INFO,←'Value1' 'VT_PTR TO VT_I4'
F.Dual.SetEventInfo 'ChangeValue1' INFO
```

If the host application was Visual Basic, a suitable callback function might be:

```
Private Sub Dual1_ChangeValue1(Value1 As Long)
Value1=2*(Value1\2)
End Sub
```

This callback function receives the proposed new value of the control as the parameter Value1, and modifies it, forcing it to be an even number.

6.529 SetFinishText Method 366

Applies To: [Section 6.463](#)

Description

The SetFinishText method sets the caption of the Finish button in a Wizard-style [Section 6.463](#).

The argument to SetFinishText is a single item as follows:

[1]	Finish button text	character vector

6.530 SetFnInfo Method 545

Applies To: [Section 6.7](#), [Section 6.420](#)

Description

This method is used to describe an APL function that is to be exported as a method, a Property Get Function, or a Property Put Function of an [Section 6.7](#) or [Section 6.420](#) object.

An exported function must be a niladic or monadic defined function (dfns and derived functions are not allowed) and may optionally return a result. Ambivalent functions (functions with optional left argument) are allowed, but will be called monadically by the host application.

COM syntax differs from APL syntax in many ways and the SetFnInfo method is required to declare an APL function to COM in terms that COM understands. In particular, although monadic APL functions take just one argument, COM functions may take several parameters, and some may be optional.

A function exported by SetFnInfo will be called by a host application with the number of parameters that SetFnInfo has described. The argument received when the function is called by a host application, will be a nested vector of this length.

The argument to SetFnInfo is a 2, 3, 4, 5 or 6-element array as follows:

[1]	Function name	character vector
[2]	Function info	nested array (see below)
[3]	Help ID	integer
[4]	Function type	integer
[5]	Property name	character vector
[6]	DISPID	integer. See DISPID (Dispatch ID)

Function info

This specifies an optional help string which describes what the function does, the data type of the result (if any) and the names and data types of its arguments.

If the function syntax is fully described, each element of *Function Info* is a 2-element vector of character vectors. The first element contains the help string and the [Section 8.2](#) of the function's result. Subsequent elements contain the name and [Section 8.2](#) of each parameter.

However, both the help string and the names of the parameters are optional and may be omitted. If so, one or more elements of *Function Info* may be a simple character vector.

Consider a very basic function ADD in an [Section 6.7](#) called `F.dbase`, that is designed to add a record to a personnel database. The database consists only of a list of names, ages and addresses.

Function ADD expects to be called with a name (character string), age (number) and address (character string), and returns a result 0 or 1 (Boolean) according to whether the record was successfully added. This function could be declared as follows:

```

HELP←'Adds a new record to the personnel database'
SPEC←(HELP 'VT_BOOL')      A Result is Boolean
SPEC,←('Name' 'VT_BSTR')   A 1st param called 'Name' is a
string
SPEC,←('Age' 'VT_I4')      A 2nd param called 'Age' is an
integer
SPEC,←('Address' 'VT_BSTR')A 3rd param called 'Address' is a
string

F.dbase.SetFnInfo 'ADD' SPEC

```

Alternatively, but much less helpfully, the function could be declared to take a single unnamed nested argument, leaving it to the host application programmer to guess at its structure :

```

SPEC←(' ' 'VT_BOOL')      A No help string, result is
Boolean
SPEC,←(' ' 'VT_ARRAY OF VT_VARIANT') A Param is a nested array
F.dbase.SetFnInfo 'ADD' SPEC

```

Help ID

This is an integer value that identifies the help context id within the help file associated with the HelpFile property of the [Section 6.7](#) object. The value -1 means that no help is provided. APL stores this information in the registry from where it may be retrieved by the host application.

Function type

This specifies the type of function being exported. This is an integer with one of the following values:

1	Function is a method
2	Function is a property get function
4	Function is a property put function

In both these last two cases, the name of the property, which is totally independent of the name of the APL function, is given as *Property name*.

If omitted, the function type is *method*.

6.531 SetItemImage Method 315

Applies To: [Section 6.629](#)

Description

This method is used to allocate a picture icon to a particular item in a [Section 6.629](#) object.

The argument to SetItemImage is a 2-element array as follows:

[1]	Item number	Integer.
[2]	Picture index	Integer.

Item number is the index of the item concerned.

Picture index is an index into the array of bitmapped images in the corresponding ImageList object which is referenced via the ImageListObj property.

6.532 SetItemPosition Event 322

Applies To: [Section 6.348](#)

Description

If enabled, this event is reported when the user drag-drops an item within a [Section 6.348](#) object. This operation may be disabled by returning 0 from a callback function.

The event message reported as the result of `WM_DROPFILES`, or supplied as the right argument to your callback function, is a 7-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'SetItemPosition' or 322
[3]	Item number	Integer. The index of the item.
[4]	Y-position	Integer. New y-position of the item.
[5]	X-position	Integer. New x-position of the item.
[6]	Button number	Integer. The mouse button used to perform the drag.
[7]	Shift State	Integer: Sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down 4 = Alt key is down

6.533 SetItemState Method 307

Applies To: [Section 6.348](#), [Section 6.629](#)

Description

This method is used to set the status of a particular item in a [Section 6.348](#) or [Section 6.629](#) object.

The argument to SetItemState is a 2-element array as follows:

[1]	Item number	Integer. The index of the item concerned.
[2]	Status	Integer

The status of an item is calculated as the sum of one or more of the following state codes:

-1	Error (most likely the Item number is invalid)
1	Item has the focus
2	Item is selected
8	Item is highlighted for dropping
16	Item is displayed in bold text
32	Item is expanded
64	Item is or has been expanded
4096	Item is checked. See CheckBoxes

6.534 SetMethodInfo Method 556

Applies To: [Section 6.412](#), [Section 6.414](#)

Description

This method is used to redefine the arguments or data types associated with a method that is exported by a COM object. SetMethodInfo is used to override the information provided by the object's Type Library.

The argument to SetMethodInfo is a 2 or 3-element array as follows:

[1]	Method name	character vector
[2]	Method info	nested vector (see below)
[3]	Method index	integer

If you wish to describe the method completely, the structure of *Method info* should be identical to the structure returned by GetMethodInfo, although abbreviated formats are also allowed.

If the object exports the method directly, and not through the standard IDispatch interface, you must also specify *Method Index*, which is the index of the method in the object's virtual table (vtable). This information may be available in printed documentation or in a C-language header file.

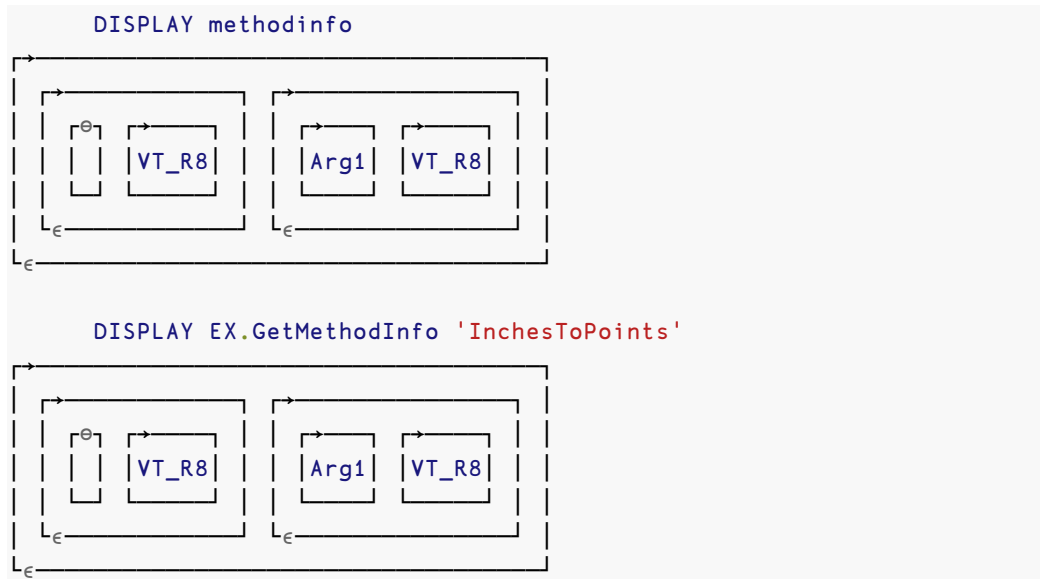
For example, the InchesToPoint method exported by *Excel.Application* takes a single argument whose name is *Arg1* and whose data type is VT_R8. The function returns a result of the same data type. The details provided in the *Excel.Application* Type Library

are in fact correct, but if you wanted to redefine them, the following statements could be used to describe the InchesToPoints method.

```
'EX' DWC 'OLEClient' 'Excel.Application' ('AutoBrowse' 0)

methodinfo← ('' 'VT_R8')('Arg1' 'VT_R8')
EX.SetMethodInfo 'InchesToPoints' methodinfo
```

Note that the structure of variable `methodinfo` is identical to the result of the `GetMethodInfo` method.



Unless you are going to call the method using the names of its arguments, these names are clearly superfluous and may be omitted, for example:

```
methodinfo← 'VT_R8' 'VT_R8'
EX.SetMethodInfo 'InchesToPoints' methodinfo
```

6.535 SetPropertyInfo Method 554

Applies To: [Section 6.412](#), [Section 6.414](#)

Description

This method is used to redefine a property that is exported by a COM object. SetPropertyInfo is used to override the information provided by the object's Type Library.

The argument to SetPropertyInfo is a 2 or 3-element array as follows:

[1]	Property name	character vector
[2]	Property info	nested vector
[3]	Property function	integer
[4]	DISPID	integer. See DISPID (Dispatch ID)

For example, the Visible property exported by *Excel.Application* has the data type VT_BOOL and may be declared as follows:

```
'EX' [WC 'OLEClient' 'Excel.Application'
EX.SetPropertyInfo 'Visible' 'VT_BOOL'
```

Property function may be required if the property value is retrieved or set via a function. This typically applies if the property takes parameters and will result in the property being fixed as a function rather than as a variable. Such properties may have a PropertyGet function, a PropertyPut function and/or a PropertyPutByReference function. If so, it is necessary to say to which of these three functions the details apply. The value of *Property function* is an integer 2 (PropertyGet), 4 (PropertyPut), or 8 (PropertyPutByReference).

For example, the following statement declares the PropertyGet function for the Item property of the Fields collection of the OLE object DAO.DBEngine. This property takes an index (into the collection) and returns an object.

```
Fields.SetPropertyInfo 'Item'('VT_DISPATCH' 'VT_I4')2
```

6.536 SetRowSize Event 175

Applies To: [Section 6.263](#)

Description

If enabled, this event is reported when the user changes the height of a row or changes the height of the column titles. This may be done by dragging a border with the mouse or by double-clicking over a border. In the former case, the default action is to adjust the height of the appropriate row or the height of the column title area to the size selected by the user. In the latter case, the default action is to adjust the height to the maximum required to display all the data.

In either case, you can disable the default action by setting the event action code to `-1` or you can selectively prevent a particular resize operation from taking place by returning 0 from a callback function.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'SelRowSize' or 175
[3]	Row number	Integer. This is sensitive to the index origin, <code>⌈IO</code> , but is <code>-1</code> if the user has resized the column titles.
[4]	Height	Integer containing the value of the (new) row height. This is <code>-3</code> if the user has double-clicked to request automatic height adjustment.
[5]	Undo flag	0 or 1

You can resize a row or resize the column titles under program control by calling `SetRowSize` as a method. If you specify `-1` as the *Height* parameter, the row will be resized to its default height .. If you specify a value of `-2` the row will be resized to fit the data. The following expression will set the heights of first `NROWS` rows of a Grid called `F.G` to fit the data and the row titles.

```
{F.G.SetRowSize w -3}''iNROWS
```

The Undo flag is always 1 if the event was generated by the user.

6.537 SetServiceState Method 93

Applies To: [Section 6.489](#)

Description

This method is used to set the state of a Dyalog APL service running under Windows. See *Installation/Configuration: Apl Application As A Service*.

The argument to the `SetServiceState` method is the desired state of the service. This should be one of the following numeric values:

SERVICE_STOPPED	1
SERVICE_RUNNING	4
SERVICE_PAUSED	7

The state of a Windows service is determined by the user and the Windows Service Control Manager (SCM).

When the SCM requests a change of state, the APL interpreter responds by setting its state to the corresponding *pending* level (SERVICE_STOPPED_PENDING, SERVICE_RUNNING_PENDING or SERVICE_PAUSED_PENDING) and then generates a [Section 6.524](#) event.

It is the responsibility of the APL service to process this event, perform the appropriate application tasks, and then respond (to the SCM) by calling the [Section 6.537](#) method to confirm that the service has reached the desired state.

6.538 SetSpinnerText Event 421

Applies To: [Section 6.563](#)

Description

If enabled, this event is generated when the user clicks one of the spin buttons in a [Section 6.563](#) object. The event is reported *after* the value of the Thumb property has been updated but *before* the Text property has been changed. You may use this event to set the text in the [Section 6.563](#) dynamically instead of relying on it being updated automatically.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'SetSpinnerText' or 421
[3]	Thumb value	Integer. The new value of the Thumb property resulting from the user pressing one of the spin buttons.
[4]	Text	The text that is about to be put into the edit field.

The SetSpinnerText event is designed to allow you to dynamically set the text in the Spinner in response to a spin button. It might be used in circumstances where the set of items you wish to present to your user is not predictable in advance.

6.539 Setup Method 101

Applies To: [Section 6.455](#)

Description

This method causes the system to display a standard [Section 6.455](#) Setup dialog box and thereby allows the user to alter the printer settings. This is a "modal" dialog box that must be closed before the APL application can continue.

The Setup method is niladic.

If you attach a callback function to this event and have it return a value of 0, the dialog box will not appear.

6.540 SetVarInfo Method 546

Applies To: [Section 6.7](#), [Section 6.420](#)

Description

This method is used to describe an APL variable that is to be exported as a property of an [Section 6.7](#) or [Section 6.420](#) object.

The argument to SetVarInfo is a 2 or 3-element array as follows:

Variable info is either a simple character vector that specifies the [Section 8.2](#) of the variable, or a 2-element vector of character vectors whose first element specifies a help string and whose second element specifies the [Section 8.2](#).

Help ID is an optional integer value that identifies the help context id within the help file associated with the HelpFile property of the ActiveXControl object. The value -1 means that no help is provided. APL stores this information in the registry from where it may be retrieved by the host application.

6.541 SetWizard Event 365

Applies To: [Section 6.463](#)

Description

If enabled, this event is reported when the user has clicked the Next or Back button in a [Section 6.463](#) with Style 'Wizard'. This action also generates PageNext (or PageBack) and PageDeactivate and PageActivate events. The SetWizard event is the final event to be reported as a result of this action, and is the only one that is affected by the result of a callback function. The event message reports the active/inactive state of the 3 page changing buttons (Back, Next and Finish) that should result from the action. Note that the Next and Finish buttons occupy the same position and are mutually exclusive.

The event message reported as the result of `□□□`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'SetWizard' or 365
[3]	Active state Back	0 or 1
[4]	Active state Next	0 or 1
[5]	Active state Finish	0 or 1
[6]	Finish caption	character vector

You may alter the state of the buttons by changing elements [3-5] of the event message and returning it as a result of your callback. You may also set the state of the buttons at any time by calling SetWizard as a method.

When the event is reported by `□□□`, element 6 is an empty vector. If you modify it and return it in the result of a callback, the caption of the Finish button changes accordingly and the Back and Next buttons disappear. This happens regardless of the states you specified in elements [3-5].

6.542 SetZoomLevel Method

Applies To: [Section 6.307](#)

Description

Sets the CEF ZoomLevel. The default (unzoomed) level is 0. Setting a positive value will increase the zoom, whereas setting a negative will decrease the zoom. The zoom scale is not linear; rather the effective scaling is approximately 1.2^{level} , so, setting the ZoomLevel to 1 will result in an approximate 20% size increase. ZoomLevel affects all

instances of HTMLRenderer windows; it is not possible to have different ZoomLevels for individual windows.

The argument to SetZoomLevel is a single numeric value:

[1]	ZoomLevel	Numeric

EXAMPLES

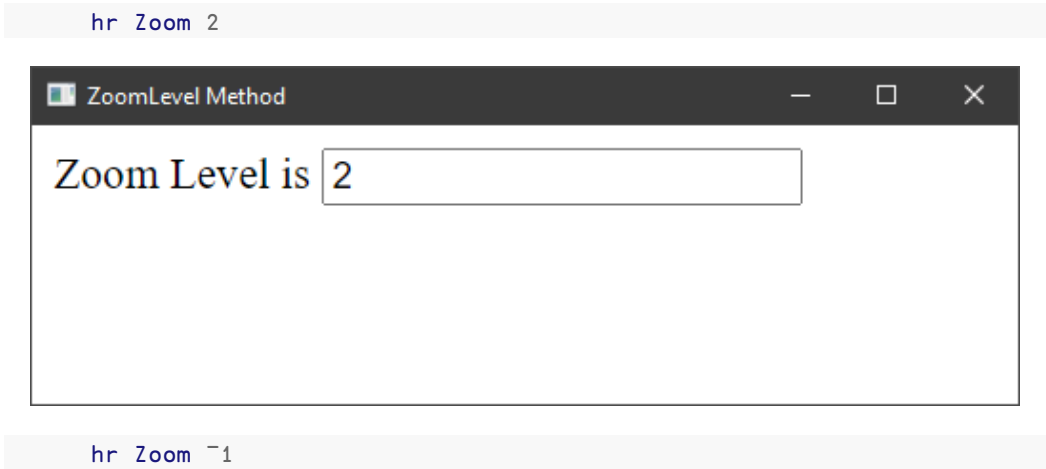
```
▼ hr Zoom level;lb;in
[1] hr.SetZoomLevel level
[2] level←hr.GetZoomLevel
[3] ((level='-')/level)←'- '
[4] lb←'<label>Zoom Level is </label>'
[5] in←'<input type="number" value="' ,level,'"></input>'
[6] hr.HTML←lb,in
▼
```

```
hr.[WC'HTMLRenderer' ('Caption' 'ZoomLevel Method')]
hr Zoom 0
```



```
hr Zoom 1
```





6.543 ShowBalloonTip Method 860

Applies To: [Section 6.581](#)

Description

The ShowBalloonTip method displays a BalloonTip in a [Section 6.581](#) object.

The argument to ShowBalloonTip is a 1, 2, 3 or 4-element array as follows:

[1]	Title	character vector
[2]	Text	character vector or matrix
[3]	Icon	Integer scalar, a character vector or a ref
[4]	Flags	Integer

The *Title* parameter is the text to be displayed in the BalloonTip title (maximum length 64).

The *Text* parameter is the text (maximum length 256) to be displayed in the BalloonTip. If omitted or empty, the BalloonTip is not displayed.

If the *Icon* parameter is an integer, it means:

0	No icon
1	Information icon
2	Warning icon
3	Error icon

Other values represent the name or a ref to an [Section 6.309](#) object. If the *Icon* parameter is omitted, no icon is displayed in the BalloonTip.

If the *Icon* parameter specifies a *large* Icon object (32 x 32 bits) the Flags parameter must be 32. Otherwise this parameter is not used.

6.544 ShowCaptions Property

Applies To: [Section 6.624](#)

Description

The ShowCaptions property specifies whether or not the captions of individual [Section 6.623](#) objects are drawn. ShowCaptions is a property of the parent [Section 6.624](#) object.

ShowCaptions is a single number with the value 0 ([Section 6.623](#) captions are not shown) or 1 (ToolButton captions **are** shown); the default is 1

ShowCaptions allows you to toggle end-user preferences for the display of [Section 6.623](#) captions, without having to set/clear individual captions one by one.

6.545 ShowComment Event 223

Applies To: [Section 6.263](#)

Description

If enabled, a [Section 6.263](#) will generate a ShowComment event when the user rests the mouse pointer over a commented cell. You may use this event to modify the appearance of the comment dynamically.

The event message reported as the result of `□□□` or supplied as the right argument to your callback function is an 8-element vector containing the following:

[1]	Object	ref or character vector
[2]	Event	' ShowComment ' or 223
[3]	Cell row	integer
[4]	Cell column	integer
[5]	Comment text	character vector
[6]	Window height	integer, pixels
[7]	Window width	integer, pixels
[8]	Tip behaviour flag	(1 = yes; 0 = no)

A callback function may modify the standard behaviour. You may prevent the comment from being displayed by returning 0 as the result of the callback. Alternatively, you may modify the comment text, its window size, or its pop-up behaviour by changing the appropriate element(s) of the event message and returning the new event message as the result.

Note that if the comment window relates to a row or column *title*, the value reported in element [3] or [4] of the event message is `-1`.

You may display the comment associated with a particular cell under program control by calling ShowComment as a method. In this case, only the *Cell row* and *Cell column* parameters need be specified. If however, you wish to override the comment text and/or its window size, you may do so (temporarily) by specifying the corresponding parameters. By default, a comment displayed under program control does not exhibit tip behaviour but remains visible until it is explicitly removed using the HideComment method.

Note that a comment will only be displayed if the specified cell is marked as a commented cell.

6.546 ShowCueWhenFocused Property

Applies To: [Section 6.56](#), [Section 6.185](#)

Description

This Boolean property specifies whether or not the text specified by the [Section 6.132](#) property should be displayed once the user has tabbed into or clicked on the empty input field (and thus given it the focus). For an [Section 6.185](#) object it applies only if the Style of the [Section 6.185](#) object is 'Single'.

6.547 ShowDevTools Method 849

Applies To: [Section 6.307](#)

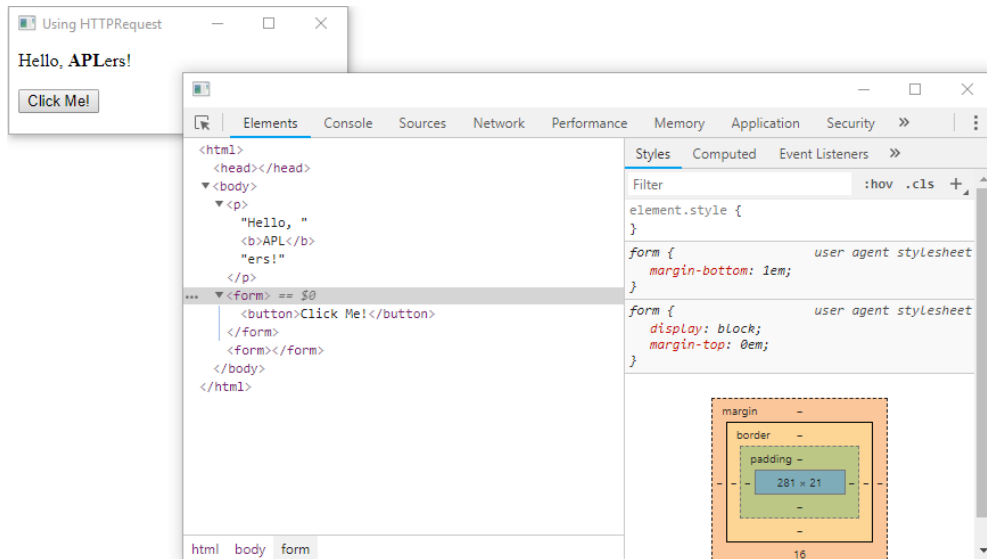
Description

This method displays or hides the CEF development tools window that displays information about the content being displayed by the HTMLRenderer.

It takes a Boolean argument 0(hide) or 1(display).

Using the example illustrated in the [HttpRequest](#) topic ...

`hr.ShowDevTools 1`



6.548 ShowDropDown Property

Applies To: [Section 6.108](#), [Section 6.624](#)

Description

The ShowDropDown property specifies whether or not a drop-down menu symbol is drawn in a [Section 6.108](#) or alongside [Section 6.623](#) objects which have [Section 6.577](#) 'DropDown'.

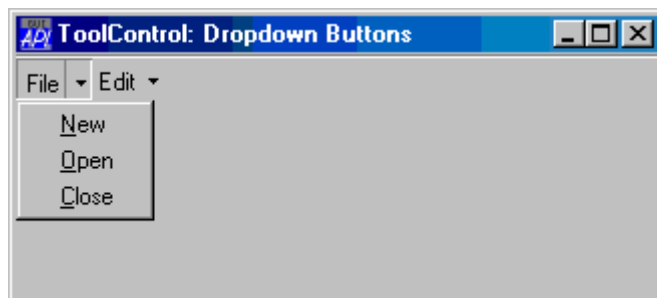
ShowDropDown is a single number with the value 0 (drop-downs captions are not shown) or 1 (drop-downs **are** shown); the default is 1.

ShowDropDown also affects the behaviour of [Section 6.623](#) objects which have Style 'DropDown'.

If the ShowDropDown property of the parent [Section 6.624](#) is 0, clicking the [Section 6.623](#) causes the popup menu to appear. In this case, the [Section 6.623](#) itself does not itself generate a Select event; you must rely on the user selecting a MenuItem to specify a particular action.

If the ShowDropDown property of the parent [Section 6.624](#) is 1, clicking the dropdown button causes the popup menu to appear; clicking the [Section 6.623](#) itself generates a Select event, but does not display the popup menu.

The following picture illustrates a [Section 6.624](#) with ShowDropDown set to 1:



6.549 ShowHelp Method 580

Applies To: [Section 6.412](#), [Section 6.414](#)

Description

This method is used to display the Windows help file for a COM object or the help topic associated with one of its properties, events or methods.

The argument to ShowHelp is $\emptyset$, or a single item as follows :

[1]	Topic	character vector.

Topic specifies the name of a property, event or method.

In the case of an OLE Control, the object name may be the name of the [Section 6.412](#) or an instance of the [Section 6.412](#).

6.550 ShowInput Property

Applies To: [Section 6.263](#)

Description

This property specifies whether or not the cells in a [Section 6.263](#) are displayed using their associated input objects.

The ShowInput property is either a single Boolean value that applies to all the cells in a [Section 6.263](#), or it is a vector whose elements are mapped to individual cells via the [Section 6.82](#) property. A value of 0 means that the corresponding cell is displayed normally. A value of 1 indicates that the cell is displayed using its associated input object, as it is when it is the current cell. ShowInput is relevant to cells displayed using [Section 6.121](#) and [Section 6.55](#) objects.

The example below illustrates the appearance of a [Section 6.263](#) in which ShowInput is set to 0 for the Job Title column and 1 for the Region and Permanent columns.

Employee Database				
Surname	Job Title	Region	Salary	Permanent
Brown	Manager	South	\$64000.00	☒
Jones	Project Leader	South	\$43250.00	☒
Green	Consultant	South	\$45000.00	☐
Black	Programmer	East	\$30000.00	☒
White	Assistant	Central	\$40000.00	☐

The appearance of the same [Section 6.263](#) but with ShowInput set to 0 throughout is illustrated below:

Employee Database				
Surname	Job Title	Region	Salary	Permanent
Brown	Manager	South	\$64000.00	1
Jones	Project Leader	South	\$43250.00	1
Green	Consultant	South	\$45000.00	
Black	Programmer	East	\$30000.00	1
White	Assistant	Central	\$40000.00	

6.551 ShowItem Method 316

Applies To: [Section 6.629](#)

Description

This method is used to display a particular item in a [Section 6.629](#) object.

The argument to ShowItem is a single item as follows:

[1]	Item number	Integer.

Item number specifies the index of the item concerned.

In order to display the requested item, the parent item (if any) will be opened and the object will be scrolled if necessary.

6.552 ShowProperties Method 560

Applies To: [Section 6.412](#)

Description

This method is used to display the PropertySheet for *an instance of* an OLE Control. The user may then modify some or all of the properties of the control by changing values in the property sheet. This facility is intended to be used in the context of a GUI design tool but may also be useful in certain end-user applications.

The ShowProperties method is niladic.

6.553 ShowSession Property

Applies To: [Section 6.420](#)

Description

This property specifies whether or not the APL Session window is displayed when an [Section 6.420](#) object is started by an OLE client.

Its default value is 0 (hide Session).

Note that if [Section 6.507](#) is 'MultiUse', you may not in any way access the *instances* of the object that are being controlled by the client applications, even if only a single client is connected.

6.554 ShowThumb Property

Applies To: [Section 6.625](#)

Description

The ShowThumb property specifies whether or not the thumb in a [Section 6.625](#) object is visible. It is Boolean with a default value of 1 and it may be toggled on and off using ☐WS.

6.555 SingleClickExpand Property

Applies To: [Section 6.629](#)

Description

The SingleClickExpand property specifies whether or not an item in a [Section 6.629](#) control is expanded when the user selects the item.

SingleClickExpand is a single number with the value 0 (the user must select the *expand icon* to cause the item to expand) or 1 (the item is expanded when the *text* of the item is selected); the default is 0.

6.556 Size Property

Applies To

Description

This is a 2-element numeric vector specifying the height and width of the object.

For the [Section 6.47](#) object, Size is set and reported in pixels. Setting the Size of a [Section 6.47](#) causes it to be scaled (up or down).

For all other objects, Size is reported and set in units defined by the [Section 6.128](#) property and, if [Section 6.128](#) is 'User', the [Section 6.661](#) and [Section 6.663](#) properties of the object's parent.

For the [Section 6.489](#) object, if [Section 6.128](#) is 'Prop' the value of Size is (100,100). If [Section 6.128](#) is 'Pixel' the value of Size reports the number of pixels on the screen.

For a [Section 6.227](#) or [Section 6.578](#), the Size property defines the area within the object, and excludes its title bar, menu bar and border if these are present.

For a [Section 6.121](#) object with a "drop-down" list, the first element of Size (height) is ignored. The height of the edit field is determined by the height of the font, while the size of the list box is determined by the [Section 6.494](#) property.

Otherwise the Size property defines the total size of the object, including borders, edges etc.

When specifying Size, you can set the height or width to a default value (□wc) or leave it unchanged (□ws) by giving the corresponding element a value of 0.

6.557 Sizeable Property

Applies To

Description

This property determines whether or not an object can be directly resized by the user once it has been created by *Dyalog APL Language: Wc*.

It is a single number with the value 0 (the object cannot be resized by the user) or 1 (the object may be resized by the user). The default is 1.

For a [Section 6.227](#), [Section 6.307](#) or [Section 6.578](#), the Sizeable property may only be set by *Dyalog APL Language: Wc* and cannot subsequently be altered using *Dyalog APL Language: Ws*. An attempt to do so generates a NONCE ERROR. For a [Section 6.227](#) or [Section 6.307](#), the default value is 1 and the object occupies a standard resizable window with a border. Note that the value of Sizeable is independent of the values of

the [Section 6.369](#) and [Section 6.388](#) properties, so that a [Section 6.227](#) or [Section 6.307](#) with [Section 6.369](#) 1 can be maximised even though its Sizeable property is 0.

For other objects, the default value of the Sizeable property is 0. However, setting it to 1 (which may be done dynamically using *Dyalog APL Language: Ws*) allows the user to resize it with the mouse.

In all these cases, when the user resizes an object, the object will generate a [Section 6.123](#) (31) event.

Sizeable also applies to the [Section 6.355](#) object. In this case, a value of 1 implies "rubberbanding" and a value of 0 means "no rubberbanding". See [Section 6.355](#) object for further details.

6.558 SM Object

Purpose: Defines a window for `□SM` / `□SR` .

Parents Form Group PropertyPage SubForm ToolBar ToolControl

Children Cursor Timer

Properties Type Posn Size Coord Border Visible Event Sizeable Draggable BCol Picture CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj AcceptFiles KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create MouseDown MouseUp MouseDbClick MouseEnter MouseLeave Configure Help DragDrop

Description

This object defines a window for `□SM`/`□SR` and allows you to combine the functionality of `□SM`/`□SR` with the "windows" GUI. For example, you can define a [Section 6.227](#) with a [Section 6.383](#) at the top and a `□SM` window beneath it, with perhaps some [Section 6.55s](#) alongside.

To allow the user to interact with both `□SM` and other top-level objects, you must specify the names of these objects in the right argument of `□SR`. Thus the statement:

```
CTX ← KEYS CTX □SR 1 2 3 'Form1'
```

allows the user to interact with fields (rows) 1-3 of `□SM` **and** with the object `'Form1'` and its children. Callback functions associated with events in `'Form1'` will be executed automatically by `□SR`. If an enabled event without a callback occurs, the event will be placed on *Dyalog APL Language: Dq's* internal queue and `□SR` will terminate. The nature of the termination (that is, that it was caused by an event in an object) is reported by the value 131072 (2×17) in the fourth element of `□SR`'s result. The specific event ([Section 6.123](#), [Section 6.396](#), etc.) is however not reported. It is therefore generally preferable to use callbacks.

The [Section 6.452](#), [Section 6.556](#) and [Section 6.128](#) properties allow you to specify the position and size of the window occupied by `□SM` within its parent [Section 6.227](#). Note however that the `□SM` window will automatically be sized to be an exact number of characters in height and width which will be reported in `□SD`.

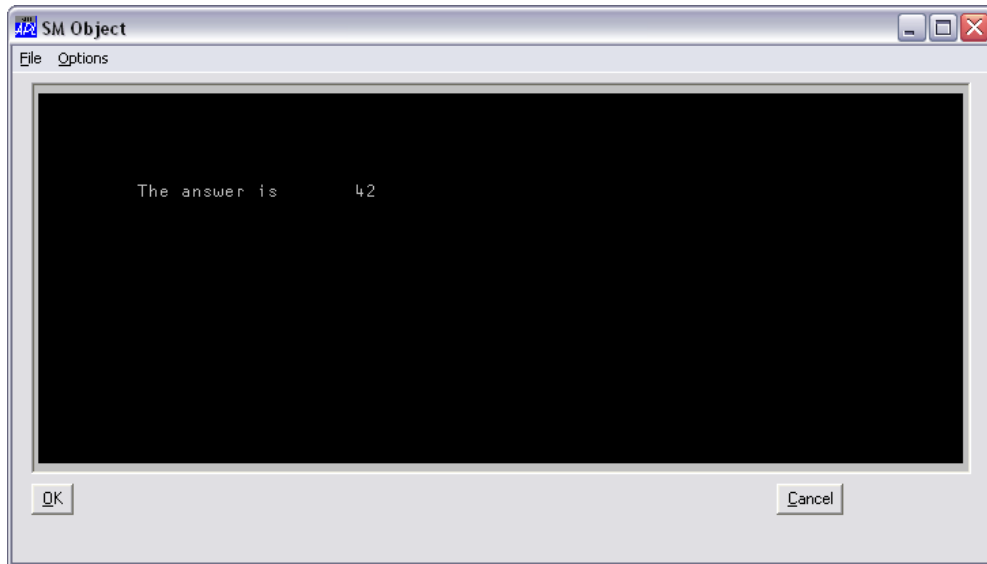
The [Section 6.49](#) property may be used to specify a border around the outside of the `□SM` window. It is a number with the value 0 (no border) or 1 (1 pixel border). The default is 0. The [Section 6.184](#) property may be used to give the object a 3-dimensional appearance. Its default value is `'Recess'`. The area within the SM object that is defined by `□SM` is necessarily a multiple of the character size. The region between this area and the outer edges of the object is coloured by the background colour specified by [Section 6.45](#), or may be filled with a bitmap specified by [Section 6.447](#).

If the user resizes the [Section 6.227](#) which contains the SM object, the SM object will generate a [Section 6.123](#) event if enabled. If the [Section 6.123](#) event is not enabled, `□SR` will terminate with a RESIZE error which can be trapped using `□TRAP`. Either method can be used to reformat `□SM` as appropriate.

The [Section 6.392](#) event can be used to bring up a pop-up menu. Note however that mouse events are not reported over `□SM` fields because `□SR` uses these to position the cursor.

The illustration shown below was produced as follows :

```
'TEST'      □WC 'Form' 'SM Object' (60 10)(40 50)
'TEST.MB'   □WC 'MenuBar'
'TEST.MB.F' □WC 'Menu' '&File'
'TEST.MB.O' □WC 'Menu' '&Options'
'TEST.B1'   □WC 'Button' '&OK' (84 2)
'TEST.B2'   □WC 'Button' '&Cancel' (84 78)
'TEST.S'    □WC 'SM'(2 2)(80 96)('BCol'192 192 192)
□SM←↑('The answer is' 5 10)(42 5 30)
```



6.559 SocketNumber Property

Applies To: [Section 6.604](#)

Description

The SocketNumber property is an integer whose value is the Window handle of the socket attached to the [Section 6.604](#) object and is generally a read-only property.

The only time that SocketNumber may be specified is when a server replicates (clones) a listening socket to which a client has just connected.

6.560 SocketType Property

Applies To: [Section 6.604](#)

Description

The SocketType property is a character vector that specifies the type of the TCP/IP socket. This is either [Section 8.8](#) (which is the default), or [Section 8.10](#).

SocketType must be defined when the object is created and may not be set or changed using `WS`.

For two Dyalog applications to communicate, their [Section 6.604](#) objects must have the same SocketType.

6.561 SortItems Property

Applies To: [Section 6.346](#)

Description

The SortItems property specifies whether or not the items in a [Section 6.346](#) object are sorted.

It is Boolean with a default value of 0. If SortItems is 1, the items are automatically sorted in alphabetical order and the object provides word recognition capabilities for selecting an item from the keyboard.

Note that the value of the [Section 6.334](#) property reflects the order of the items displayed in the [Section 6.346](#) object rather than the order in the array that was used to assign it.

This property may only be initialised when the object is created and cannot subsequently be changed.

6.562 Spin Event 420

Applies To: [Section 6.563](#), [Section 6.637](#)

Description

If enabled, this event is generated when the user clicks one of the spin buttons in a [Section 6.563](#) object. The event is reported *before* the value of the Thumb property has been updated. You may disable the operation of the spin buttons by disabling this event. You may selectively prevent the user selecting a particular value by returning 0 from a callback function. You may also return a modified event message as a result in order to set the Thumb property to a different value.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Spin' or 420
[3]	Thumb value	Integer. The new value of the Thumb property resulting from the user pressing one of the spin buttons.
[4]	Adjustment	Integer. The amount by which the new value of the Thumb differs from its previous value.

6.563 Spinner Object

Purpose: The Spinner object allows the user to enter a value, using an [Section 6.637](#) object to adjust it as required.

Parents ActiveXControl CoolBand Form Grid Group PropertyPage SubForm ToolBar ToolControl

Children Circle Ellipse Font Marker Poly Rect Text Timer

Properties Type Items Text Posn Size Coord Align Border Justify Active Visible Event Thumb Step Wrap Limits Sizeable Dragable FontObj FCol BCol CursorObj AutoConf Data Attach EdgeStyle Handle Hint HintObj Tip TipObj FieldType MaxLength Decimals Password ValidIfEmpty ReadOnly FormatString Changed Value Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap Spin SetSpinnerText Select BadValue KeyError Change

Description

The Spinner object is a special Dyalog APL composite object that consists of an edit field and a pair of spin buttons. The user may enter a value by typing in the edit field and may adjust the value with the spin buttons. The Spinner may cycle through a pre-defined set of values specified by the [Section 6.334](#) property or through a range of values specified by the [Section 6.345](#) property. The [Section 6.206](#) property supports all of the standard data types, that is, Char, Numeric, LongNumeric, Currency, Date, LongDate, and Time.

The [Section 6.345](#) property is a 2-element numeric vector that specifies the minimum and maximum value of the object. The [Section 6.576](#) property specifies the amount by which the value is incremented or decremented by the spin buttons. The current value in the object is defined by the [Section 6.608](#) and [Section 6.641](#) properties, which are usually identical. If [Section 6.470](#) is 0, the user may type a value into the edit field which will be validated and converted according to the [Section 6.206](#). In this case, the [Section 6.641](#) and the [Section 6.608](#) properties may be different.

An *alternative* way to use the Spinner object is to specify the [Section 6.334](#) property. This defines a discrete set of values through which the user may cycle, and the object behaves rather like a [Section 6.121](#) without a drop-down list. In this case, the [Section 6.345](#) property is automatically set to `(1, nItems)`, [Section 6.608](#) refers to the *index* into the list of [Section 6.334](#), and [Section 6.576](#) specifies the amount by which this index is updated by the spin buttons. For example, if you set [Section 6.576](#) to 3, the spin buttons would display every third item. The [Section 6.334](#) property may be a character matrix, a vector of character vectors, or a numeric vector and will be formatted according to the [Section 6.206](#). For example, if you wanted the user to select one of a set of specific dates, you would set the [Section 6.206](#) to Date or LongDate and the [Section 6.334](#) property to the day numbers (since 1 January 1900) corresponding to the dates you require. The [Section 6.470](#) property specifies whether or not the user may enter data into the edit field. A value typed in by the user will be converted and formatted according to the [Section 6.206](#) but need not correspond to a value in Items.

In operation, the value in the Spinner is adjusted by the [Section 6.576](#) each time one of the spin buttons is clicked. If the user holds a spin button down, the value is adjusted at the rate defined for the keyboard repeat rate. Furthermore, the size of each adjustment is increased according to the length of time the button stays depressed. After 1 second, the amount is increased to $(2 \times \text{Step})$, after 2 seconds, to $(4 \times \text{Step})$, after 3 seconds to $(8 \times \text{Step})$ and so forth until the amount of adjustment exceeds one quarter of the range $(\text{Limits}[2] \pm \text{Limits}[1])$.

When the value in the spinner reaches its top or bottom limit, it will wrap around to the opposite limit if the value of the [Section 6.660](#) property is 1 (the default). Otherwise it will stick.

The [Section 6.371](#) property defines the maximum number of characters that the user may type into the edit field. The [Section 6.147](#) property specifies the number of decimal places to which a numeric value is displayed and applies only if the [Section 6.206](#) is Numeric or LongNumeric.

The Spinner generates two special events, [Section 6.562](#) and [Section 6.538](#). The [Section 6.562](#) event is generated each time the value of the [Section 6.608](#) is about to be updated and reports the new value and the difference between it and the current

value. You may prevent the [Section 6.608](#) from being updated by returning 0 from a callback function, or you may alter the new value of the [Section 6.608](#) by returning a modified message. The [Section 6.538](#) event is generated after the [Section 6.608](#) has been reset but *before* the edit field has been updated. It reports the new value of the [Section 6.608](#) and the text that is about to be written into the edit field. By returning a modified event message from a callback, this event allows your application to respond *dynamically* to the spin buttons and to control the text in the edit field directly.

Like an [Section 6.185](#) object, the Spinner has a [Section 6.86](#) property and generates a Change event when loses the focus after the value of its Text and/or [Section 6.608](#) property has been altered.

If [Section 6.206](#) is Numeric, LongNumeric, Currency, Date, LongDate or Time, the Spinner will generate a [Section 6.38](#) event when it loses the focus if the text in the edit field (that is, the Text property) is in conflict with the [Section 6.206](#) property and cannot be converted to an appropriate number, or is outside the range specified by the [Section 6.345](#) property. If the edit field is empty, a [Section 6.38](#) event will be generated if [Section 6.639](#) is 0, but not if it is set to 1.

6.564 SplitObj1 Property

Applies To: [Section 6.566](#)

Description

The SplitObj1 property specifies the name of, or ref to, one of up to two objects managed by a [Section 6.566](#) object. The object must be one of the following types:

Button	Calendar	Combo	Edit	Grid	Group
Label	List	ListView	MDIClient	ProgressBar	RichEdit
Scroll	Spinner	Static	StatusBar	SubForm	TabBar
TabControl	ToolBar	TrackBar	TreeView	UpDown	

If the [Section 6.577](#) property of the [Section 6.566](#) is 'Vert', the object specified by SplitObj1 is positioned at (0 0) and sized to occupy the space in its parent to the left of the [Section 6.566](#), with the [Section 6.566](#) itself attached to its right edge.

If the [Section 6.577](#) property of the [Section 6.566](#) is 'Horz', the object specified by SplitObj1 is positioned at (0 0) and sized to occupy the space in its parent above the [Section 6.566](#), with the [Section 6.566](#) itself attached to its bottom edge.

If SplitObj1 is empty, the [Section 6.566](#) manages the single object specified by [Section 6.565](#) and the space to the left or above the [Section 6.566](#) is empty or controlled by another [Section 6.566](#).

6.565 SplitObj2 Property

Applies To: [Section 6.566](#)

Description

The SplitObj2 property specifies the name of, or ref to, one of up to two objects managed by a [Section 6.566](#) object. The object must be one of the following types:

Button	Calendar	Combo	Edit	Grid	Group
Label	List	ListView	MDIClient	ProgressBar	RichEdit
Scroll	Spinner	Static	StatusBar	SubForm	TabBar
TabControl	ToolBar	TrackBar	TreeView	UpDown	

If the [Section 6.577](#) property of the [Section 6.566](#) is 'Vert', the object specified by SplitObj2 is initially positioned at (0 x), where x is half the width of the parent plus the Size of the [Section 6.566](#), and sized to occupy the space in its parent to the right of the [Section 6.566](#), with the [Section 6.566](#) itself attached to its left edge.

If the [Section 6.577](#) property of the [Section 6.566](#) is 'Horz', the object specified by SplitObj2 is initially positioned at (y 0), where y is half the height of the parent plus the Size of the [Section 6.566](#), and sized to occupy the space in its parent below the [Section 6.566](#), with the [Section 6.566](#) itself attached to its top edge.

If SplitObj2 is empty, the [Section 6.566](#) manages the single object specified by [Section 6.564](#) and the space to the right or below the [Section 6.566](#) is empty or controlled by a second [Section 6.566](#).

6.566 Splitter Object

Purpose: The Splitter object divides a container into resizable panes.

Parents ActiveXControl Form Group PropertyPage SubForm

Children Timer

Properties Type SplitObj1 SplitObj2 Posn Size Style Coord Align Active Visible Event
BCol CursorObj Data KeepOnClose MethodList ChildList EventList PropList

Methods Detach

Events Close Create StartSplit Splitting EndSplit

Description

The Splitter divides the client area of a [Section 6.227](#) or [Section 6.578](#) into resizable panes. Each pane created this way may be empty or be occupied by a single object. If the object in a pane is itself a container object, such as a [Section 6.578](#), it may have a number of other controls within it.

A single Splitter may manage the geometry of 0, 1 or 2 other objects, which, together with the Splitter itself, share the same parent. The two objects are named by the [Section 6.564](#) and [Section 6.565](#) properties respectively.

A Splitter may manage objects of the following types:

Button	Calendar	Combo	Edit	Grid	Group
Label	List	ListView	MDIClient	ProgressBar	RichEdit
Scroll	Spinner	Static	StatusBar	SubForm	TabBar
TabControl	ToolBar	TrackBar	TreeView	UpDown	

If [Section 6.577](#) is 'Vert' (the default), the Splitter is drawn vertically in its parent with the first object ([Section 6.564](#)) positioned to its left, and the second object ([Section 6.565](#)) to its right. [Section 7.6](#)

If [Section 6.577](#) is 'Horz', the Splitter is drawn horizontally in its parent with the first object ([Section 6.564](#)) positioned above, and the second object ([Section 6.565](#)) below. [Section 7.7](#)

The [Section 6.577](#) property must be set when the object is created with `□WC` and may not subsequently be changed using `□WS`.

The [Section 6.452](#) and [Section 6.556](#) properties are partially read-only, in that only one dimension of the value may be specified. If [Section 6.577](#) is 'Vert', you may specify the x-coordinate and the width of the Splitter, but you may not specify its y-coordinate nor its height. If [Section 6.577](#) is 'Horz', you may specify the y-coordinate and the width of the Splitter, but you may not specify its x-coordinate nor its length.

When the user positions the mouse pointer directly over the Splitter object, the cursor changes (by default) to a double-headed arrow (direction in accordance with [Section 6.577](#)). The user may now depress the left mouse button and drag the Splitter to a new position, resizing the objects named by [Section 6.564](#) and [Section 6.565](#) in the process.

You can select a different cursor using the `CursorObj` property. Note that setting the `CursorObj` property to 0 selects the default cursor, which is the appropriate double-headed arrow.

When the user depresses the mouse button, the Splitter generates a [Section 6.570](#) event. When the user releases the mouse button, the Splitter generates an [Section 6.194](#) event. If full-drag is in effect, the Splitter also reports [Section 6.567](#) events as it is dragged. All these events report the new or current position of the Splitter object and are provided for information only.

Note that the objects named by [Section 6.564](#) and [Section 6.565](#) and any sub-objects they contain will generate `Configure` events when they are resized by the Splitter.

Alignment

The [Section 6.13](#) property specifies how a Splitter behaves when its parent is resized and may be 'None', 'Left', 'Right', 'Top' or 'Bottom'.

If [Section 6.13](#) is 'None', the Splitter moves as its parent is resized, so that it divides its parent in the same *proportions* as before. This is the default.

Any other value of [Section 6.13](#) attaches the Splitter to the corresponding edge of its parent. For example, if [Section 6.13](#) is 'Left', the width of the object to the *left* of the Splitter remains fixed when its parent is resized horizontally by the user.

Like the [Section 6.577](#) property, [Section 6.13](#) may be set only when the object is created with `WC` and may not subsequently be changed using `WS`.

Using Multiple Splitters

If you want to divide a Form into more than 2 resizable panes, there are two possible approaches, each with its own different characteristics.

The first approach is a hierarchical one using SubForms. This example shows how you can create a Form containing three resizable Edit objects.

First, you create an Edit, a SubForm, and a Splitter as children of the *Form*, using the Splitter to divide the Form into two panes, one for the Edit and the other for the

SubForm. Next, you create two Edit objects and a Splitter as children of the *SubForm*, using the second Splitter to divide the SubForm into two. You can continue with this approach to any reasonable depth. [Section 7.8](#)

Notice that, by default, when the first Splitter is shifted to the left, both panes in the SubForm expand equally.

The second approach is to create multiple Splitters *at the same level*, that is, owned by the same parent. [Section 7.9](#)

In this case, the third Edit object *F . E3* is unaffected by movement of the leftmost Splitter *F . S1*. Note also, that the first Splitter *F . S1* may not be dragged further right than the second Splitter *F . S2*.

Using the non-hierarchical approach, horizontal and vertical Splitters may be combined in interesting ways. This can also be achieved using nested SubForms, but at the expense of a complex object hierarchy. [Section 7.10](#)

Notice that in this example, with the exception of the last Splitter *F . S4*, it is necessary only to specify the [Section 6.564](#) property for each of the Splitters. The reason is that the first four Splitters only manage one object *directly*. For example, the object to the right of *F . S1* is in fact a horizontal Splitter *F . S2*. Dragging *F . S1* changes the length of *F . S2* which in turn changes the width of *F . E2*. and *F . E3*.

Colliding Splitters

If you have two or more vertical Splitters or two or more horizontal Splitters in the same parent object, it is possible for the user to make the Splitters *collide*. This can occur by dragging one of the Splitters into the other, or, unless both Splitters have Align set to 'None', by shrinking the parent.

When Splitters collide, the object being dragged by the user (a Splitter or a border of the parent) takes precedence over the setting of Align, and temporarily *pushes* other Splitters along in its direction of travel. If and when the operation is reversed, the other Splitters are *pulled* back to their original positions.

6.567 Splitting Event 281

Applies To: [Section 6.566](#)

Description

If enabled, this event is reported while a [Section 6.566](#) object is being dragged, between a [Section 6.570](#) and an [Section 6.194](#). This event is only reported if full-drag is enabled.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to -1 or by returning 0 from a callback function.

The event message reported as the result of `□□□□□□□□`, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Splitting' or 281
[3]	Y	y-position of top left corner
[4]	X	x-position of top left corner
[5]	H	height of the Splitter
[6]	W	width of the Splitter

See also [Section 6.570](#), [Section 6.194](#).

6.568 Start Property

Applies To: [Section 6.94](#), [Section 6.190](#)

Description

This property specifies one or more start-angles for an arc, pie-slice, or chord of a circle or ellipse. It may be used in conjunction with [Section 6.192](#) which specifies end angles. Angles are measured counter-clockwise from the x-axis at the centre of the object.

If a single arc is being drawn, Start is a single number that specifies the start angle of the arc in radians ($0 \leq \theta < 2\pi$). If multiple arcs are being drawn, Start is either a single number as before (the start angle for several concentric arcs) or a numeric vector with one element per arc.

If [Section 6.192](#) is not specified, the default value of Start is 0. Otherwise, the default value of Start is (0 , -1 ↑ + \End).

6.569 StartIn Property

Applies To: [Section 6.51](#)

Description

The StartIn property is a character string that specifies the start point and root for a [Section 6.51](#) object.

Only the specified folder and its subfolders appear in the dialog box. The user cannot browse higher in the folder architecture than this folder.

The default value for StartIn is an empty vector which means that the root of the browse dialog is the desktop.

6.570 StartSplit Event 280

Applies To: [Section 6.566](#)

Description

If enabled, this event is reported when the user depresses the left mouse button over a [Section 6.566](#) object to signify the beginning of a drag operation.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to -1 or by returning 0 from a callback function.

The event message reported as the result of `□□□□`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'StartSplit' or 280

See also [Section 6.194](#), [Section 6.567](#).

6.571 State Property

Applies To: [Section 6.55](#), [Section 6.227](#), [Section 6.578](#), [Section 6.584](#), [Section 6.623](#)

Description

This property determines the state of a [Section 6.55](#), [Section 6.584](#), [Section 6.623](#), [Section 6.227](#), or [Section 6.578](#). It is a single number with the value 0 (the default), 1, or 2 ([Section 6.227](#) and [Section 6.578](#)).

If the [Section 6.577](#) property is 'Push', a State of 0 means that the pushbutton is displayed normally (out). If its State is 1, the pushbutton is displayed depressed (in).

If the [Section 6.577](#) property is 'Radio' or 'Check', 0 means "not selected" and 1 means "selected". Note that only one of a group of buttons with [Section 6.577](#) 'Radio' that share the same parent may have State 1. Setting State to 1 automatically deselects all the others in the group.

For a [Section 6.227](#) or [Section 6.578](#), a value of State of 0 means that the [Section 6.227](#) is currently displayed in its "normal" state. 1 means that the [Section 6.227](#) is currently minimised (displayed as an icon). The value 2 indicates that the [Section 6.227](#) is maximised and displayed full-screen. The State of a [Section 6.227](#) can be changed using *Dyalog APL Language: Ws*.

6.572 StateChange Event 35

Applies To: [Section 6.227](#), [Section 6.578](#)

Description

This event is generated by a [Section 6.227](#) or [Section 6.578](#) when the user attempts to change the [Section 6.571](#) of a [Section 6.227](#), by minimising it, maximising it, or restoring it from a minimised or maximised state. The event is reported **before** the window changes state. You may prevent the state change by disabling the event (action code -1) or by returning a 0 result from an attached callback function.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'StateChange' or 35
[3]	Window state	0 (about to be restored) 1 (about to be minimised) 2 (about to be maximised)

6.573 Static Object

Purpose: This object is primarily used to display graphics in a sub-window.

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm ToolBar
ToolControl

Children Circle Cursor Ellipse Font Image Locator Marker Metafile Poly Rect Text Timer

Properties Type Posn Size Style Coord Border Active Visible Event Sizeable Draggable
FontObj FCol BCol Picture CursorObj AutoConf YRange XRange Data Attach TextSize
EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles
KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ChooseFont

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose
Help MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave
MouseWheel FontOK FontCancel Select

Description

The overall appearance of an empty Static object is controlled by the value of its [Section 6.577](#) property which may be one of the following character vectors:

'BlackFrame'	'BlackBox'
'GreyFrame' or 'GrayFrame'	'GreyBox' or 'GrayBox'
'WhiteFrame'	'WhiteBox'

Note that the colours implied by the [Section 6.577](#) are not "hard-coded" but are actually defined by the current Windows colour scheme as follows:

Black	Window Border Colour
Grey/Gray	Desktop Colour
White	Window Background Colour

If the background colour of the [Section 6.227](#) is also set to the Window Background Colour, it follows that the [Section 6.577](#)s 'WhiteFrame' and 'WhiteBox' make the Static itself invisible (against the background), although the **contents** of the Static will show. This makes the Static appear like an invisible clipping window.

6.574 StatusBar Object

Purpose: This object is used to manage [Section 6.575](#) objects which display information for the user.

Parents ActiveXControl CoolBand Form SubForm

Children Bitmap BrowseBox Circle Cursor Ellipse FileBox Font Icon Image Marker Poly ProgressBar Rect StatusField Text Timer

Properties Type Posn Size Coord Align Border Active Visible Event VScroll HScroll Sizeable FontObj FCol BCol Picture CursorObj AutoConf YRange XRange Data Attach TextSize EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj ChooseFont

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles DropObjects Expose Help MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel Select

Description

The StatusBar is a container object that manages [Section 6.575s](#). [Section 6.575](#) objects display textual information and are typically used for help messages and for monitoring the status of an application. They can also be used to automatically report the status of the Caps Lock, Num Lock, Scroll Lock, and Insert keys.

```
'TEST'□WC'Form' 'StatusBar' ( 'EdgeStyle' 'Default' )
'TEST.SB'□WC'StatusBar '
'TEST.SB.S1'□WC'StatusField' 'Field1:' 'text1'
'TEST.SB.S2'□WC'StatusField' 'Field2:' 'text2'
'TEST.SB.S3'□WC'StatusField' 'Field3:' 'text3'
```



The [Section 6.13](#) property determines to which side of the parent [Section 6.227](#) or [Section 6.578](#) the StatusBar is attached. By default, a StatusBar is positioned along the lower edge of the [Section 6.227](#) ([Section 6.13](#) 'Bottom'). Using the [Section 6.13](#), [Section 6.452](#) and [Section 6.556](#) properties you may create StatusBars in different positions and with differing sizes if you wish. Notice that the [Section 6.13](#) property controls how the StatusBar reacts to its parent [Section 6.227](#) being resized. If [Section 6.13](#) is 'Top' or 'Bottom', the StatusBar remains fixed in height but stretches and shrinks sideways with the [Section 6.227](#). If [Section 6.13](#) is 'Left' or 'Right', the StatusBar remains fixed in width and stretches and shrinks vertically with the [Section 6.227](#).

By default a StatusBar has a Button Face colour background and the value of its [Section 6.184](#) property is 'Default'. This gives it the appearance shown above.

Unless you specify the position and size of its children, a StatusBar automatically manages their geometry. The first [Section 6.575](#) is positioned just inside its top left corner. If [Section 6.13](#) is 'Top' or 'Bottom', the next [Section 6.575](#) is positioned alongside the first but with a small gap between them. Subsequent [Section 6.575](#)s are added in a similar fashion. If [Section 6.13](#) is 'Left' or 'Right', the second and subsequent [Section 6.575](#)s are added below the first with a similar gap between them. In either case you can position and size the [Section 6.575](#)s explicitly if you wish.

If you attempt to add a [Section 6.575](#) that would extend beyond the right edge ([Section 6.13](#) 'Top' or 'Bottom') or bottom edge ([Section 6.13](#) 'Left' or 'Right') the behaviour depends upon the value of [Section 6.303](#) or [Section 6.646](#). If [Section 6.303](#) is 0 (the default) and [Section 6.13](#) is 'Top' or 'Bottom', the [Section 6.575](#) is added below the first one, thereby starting a new row. If [Section 6.646](#) is 0 (the default) and [Section 6.13](#) is 'Left' or 'Right', it is added to the right of the first one thereby starting a new column. If [Section 6.303](#) or [Section 6.646](#) is -1 or -2, the new [Section 6.575](#) is simply positioned in the same row or column and may be scrolled into view using a mini scrollbar. A value for [Section 6.303](#) or [Section 6.646](#) of -1 causes the mini scrollbar to be permanently present in the [Section 6.508](#) Bar. A value of -2 causes it to appear only when required.

[Section 6.646](#) and [Section 6.303](#) may only be set when the object is created and may not subsequently be changed.

6.575 StatusField Object

Purpose: This object is used to display information for the user.

Parents StatusBar

Children Menu Timer

Properties Type Caption Text Posn Size Style Coord Border Visible Event Sizeable
 Draggable FontObj FCol BCol Picture AutoConf Data Attach EdgeStyle Translate
 Accelerator KeepOnClose MethodList ChildList EventList PropList

Methods Detach

Events Close Create DropObjects Select MouseDown MouseUp MouseMove
 MouseDbClick

Description

The StatusField object provides an area for displaying context sensitive help messages, keyboard status, and other application dependent information.

By default a StatusField is a recessed rectangle in which information is displayed. It has a [Section 6.66](#) and a [Section 6.606](#) property, which by default are empty, but either or both of which can be used to present information. The [Section 6.66](#) is left justified in the field and the [Section 6.606](#) is displayed immediately to its right. Typically, you would use the [Section 6.66](#) property as a title to describe the information that the StatusField displays, and the [Section 6.606](#) property to show its current value. However, you are not obliged to use both of them and you can achieve most effects with just one.

Note that when the StatusField is used to display hints it is its [Section 6.606](#) property that is used.

A StatusField may be used to monitor the status of the keyboard and this is controlled by its [Section 6.577](#) property. The default value for [Section 6.577](#) is an empty vector. However, you can set it to monitor various keyboard states as follows :

CapsLock	Monitors state of Caps Lock key
ScrollLock	Monitors state of Scroll Lock key
NumLock	Monitors state of Num Lock key
KeyMode	Monitors the keyboard mode (APL/ASCII)
InsRep	Monitors the state of the Insert key

In each case, the [Section 6.606](#) property of the StatusField is used to display the keyboard status. If [Section 6.577](#) is CapsLock, ScrollLock or NumLock, the field displays "Caps", "Num" or "Scroll" if the corresponding mode is selected and is blank if not.

If [Section 6.577](#) is InsRep, the StatusField displays either "Ins" or "Rep". Initially it always displays "Ins" and then toggles between "Rep" and "Ins" each time the Insert key is pressed.

If [Section 6.577](#) is KeyMode, the StatusField displays the name for the current keyboard mode which is defined in the input table being used. For the 2-mode tables APL_US.DIN, APL_UK.DIN etc., the mode name displayed is either "Apl" or "Asc". The unified tables have no modes so a StatusField with this [Section 6.577](#) does nothing.

If [Section 6.577](#) is set to one of the above, you may still use the [Section 6.66](#) property to give the StatusField a title. You may even set the value of the [Section 6.606](#) property, but be aware that this value will be reset when the user next presses the key the StatusField is monitoring.

6.576 Step Property

Applies To

Description

This property determines the size of changes reported when the user clicks a scroll arrow (small change) or clicks on the body of the scrollbar (large change). The object's [Section 6.608](#) property increases or decreases by this amount.

For a [Section 6.508](#) object, Step is a 2-element numeric vector whose first element specifies the value of the "small change" and whose second element specifies the value of the "large change".

For a [Section 6.227](#) or [Section 6.578](#), Step is a 4-element numeric vector. The first two elements refer to the [Section 6.227](#)'s vertical scrollbar and the second two elements refer to the [Section 6.227](#)'s horizontal scrollbar.

For the above objects, values of Step must be between 1 and the value of the [Section 6.469](#) property.

For a [Section 6.355](#) object, Step is a 2-element integer vector (default value 1 1) that specifies the increments (in pixels) by which the size or position of the [Section 6.355](#) changes in the Y and X directions respectively as the user moves the [Section 6.355](#).

6.577 Style Property

Applies To

Description

This property determines a particular style of object within the general category of [Section 6.630](#).

It is a character vector whose value depends upon the type of object.

For a [Section 6.55](#), Style may be 'Push', 'Radio', 'Check', 'Toggle', 'CommandLink' or 'Split'.

'Push' specifies that the button appears and behaves like a pushbutton (sometimes also called a command button).

'Radio' means that the button is displayed as a small circle accompanied by a description. When the button is selected, the circle is filled in. In a group of buttons with Style 'Radio' that share the same parent, only one of them may be selected. This style of button is generally known as a "radio-button" or an "option button".

'Check' or 'Toggle' means that the button is displayed as a small box accompanied by a description. When the button is selected a cross appears in the box. This style of button is known as a "check-box".

'CommandLink' means that the button has an icon displayed to the left of its [Section 6.66](#), the appearance of which is controlled by the [Section 6.189](#) property. **Note that this feature only applies if Native Look and Feel

is enabled.**

'Split' specifies a 'Push' button with an additional drop-down button, similar to that provided by a [Section 6.121](#) object. **Note that this feature only applies if Native Look and Feel

is enabled.**

For a [Section 6.58](#) object, The Style property may be either 'Single' (the default) or 'Multi'. If Style is 'Single', the user may select a single date. If Style is 'Multi', the user may select a contiguous range of dates.

For a [Section 6.121](#) or [Section 6.122](#) object, Style may be 'Simple' 'DropEdit' or 'Drop' (the default). 'Simple' specifies a simple combo box in which the associated list box is displayed at all times. The other two styles provide list boxes which "drop down" when the user clicks on a symbol displayed to the right of the [Section 6.121](#)'s edit field. A 'DropEdit' Style allows the user to type (anything) in the edit field. A

'Drop' Style forces the contents of the edit field to be either empty or one of the choices specified by [Section 6.334](#).

For a [Section 6.143](#), Style may be either 'Combo' (the default) or 'UpDown'.

For an [Section 6.185](#) object, Style may be 'Single' or 'Multi'. If Style is 'Single' the object displays only a single line of text and the user may not enter any more lines. If the Style is 'Multi' the number of lines displayed is governed by the [Section 6.494](#) or [Section 6.556](#) property and the user may insert, add or delete lines as desired.

For [Section 6.208](#), [Section 6.346](#), and [Section 6.348](#) objects, Style may be 'Single' or 'Multi'. If the Style is 'Single' only one file or item can be selected. If Style is 'Multi', several files or items can be selected.

For an [Section 6.309](#), Style may be 'Large' (the default) or 'Small' and specifies the size of the icon (32x32 or 16x16) to be loaded from a file.

For a [Section 6.355](#), Style may be 'Point', 'Rect' (the default), 'Line' or 'Ellipse'. It specifies the shape that is drawn as the user moves the mouse.

For a [Section 6.384](#), Style may be 'Check' (the default) or 'Radio'. The latter specifies that within a contiguous block of such MenuItem's, only one may have [Section 6.90](#) set to 1. Setting [Section 6.90](#) to 1 on any item in that group automatically sets [Section 6.90](#) to 0 on the others. A radio style [Section 6.384](#) that is checked has a small radio dot drawn to the left of its Caption.

For a [Section 6.399](#), the Style property determines the type of icon which is displayed in it. This is a character vector with one of the following values:

'Msg'	no icon (the default)
'Info'	information message icon
'Query'	query (question) icon
'Warn'	warning icon
'Error'	critical error icon

For a [Section 6.573](#) object, Style defines its appearance, and may be one of:

'BlackFrame'	'BlackBox'
'GreyFrame' or 'GrayFrame'	'GreyBox' or 'GrayBox'
'WhiteFrame'	'WhiteBox'

A [Section 6.575](#) may be used to monitor the state of a key on the keyboard. If so, its Style property determines the key it monitors and may be one of the following:

CapsLock	Monitors state of Caps Lock key
ScrollLock	Monitors state of Scroll Lock key
NumLock	Monitors state of Num Lock key
KeyMode	Monitors the keyboard mode (APL/ASCII)
InsRep	Monitors the state of the Insert key

For a [Section 6.566](#), the Style property specifies the orientation of the [Section 6.566](#) and may be 'Vert' (the default) or 'Horz'.

For a [Section 6.585](#), the Style property determines the appearance of its [Section 6.584](#) children, and may be 'Tabs' (the default), 'Buttons' or 'FlatButton'.

For a [Section 6.604](#), Style is a character vector that specifies the type of data transmitted or received by the socket; it may be 'Char', 'Raw', or 'APL'. The value APL is valid only if the [Section 6.560](#) is 'Stream'.

For a [Section 6.623](#), the Style property specifies the behaviour of the button and may be 'Push' (the default), 'Check', 'Radio', 'DropDown', or 'Separator'.

For a [Section 6.624](#), the Style property determines the appearance of its [Section 6.623](#) children and may be 'Buttons', 'FlatButton' (the default), 'List' or 'FlatList'.

For a [Section 6.625](#), the Style property determines the appearance and behaviour of the TrackBar and may be 'Standard' (the default) or 'Selection'.

6.578 SubForm Object

Purpose: This object represents a window that is owned by and constrained within another [Section 6.227](#) or an [Section 6.378](#).

Parents ActiveXControl CoolBand Form Group MDIClient PropertyPage SubForm TabControl ToolBar ToolControl

Children Animation Bitmap BrowseBox Button ButtonEdit Calendar Circle ColorButton Combo ComboEx Cursor DateTimePicker Edit Ellipse FileBox Font Form Grid Group HTMLRenderer Icon Image ImageList Label List ListView Locator Marker MDIClient Menu MenuBar Metafile MsgBox OCXClass Poly ProgressBar PropertySheet Rect

RichEdit Scroll SM Spinner Splitter Static StatusBar SubForm TabBar TabControl
TCPSocket Text Timer TipField ToolBar ToolControl TrackBar TreeView UpDown

Properties Type Caption Posn Size Coord State Border Active Visible Event Thumb
Range Step VScroll HScroll Sizeable Moveable SysMenu MaxButton MinButton
HelpButton FontObj BCol Picture OnTop IconObj CursorObj AutoConf YRange XRange
Data Attach TextSize EdgeStyle Handle Hint HintObj Tip TipObj TabObj Translate
Accelerator AcceptFiles KeepOnClose Dockable Docked DockShowCaption
DockChildren UndocksToRoot Redraw TabIndex PageSize MethodList ChildList EventList
PropList

Methods Detach ChooseFont GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles
DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp
MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom
GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap StateChange
MDIActivate MDIDeactivate DockStart DockMove DockRequest DockAccept DockEnd
DockCancel Select FrameContextMenu VScroll HScroll VThumbDrag HThumbDrag

Description

If the SubForm is the child of a [Section 6.227](#), it is by default a simple featureless window that occupies the entire client area (excluding standard [Section 6.621s](#), [Section 6.574s](#) and [Section 6.582s](#)) of its parent.

The properties that control its appearance, including [Section 6.557](#), [Section 6.398](#), [Section 6.580](#), [Section 6.49](#), [Section 6.369](#) and [Section 6.388](#), all default to 0. The [Section 6.184](#) property also defaults to 'None', so the background of the SubForm defaults to the Window Background colour.

If the SubForm is the child of an [Section 6.378](#), its default appearance is the same as for a top-level [Section 6.227](#). By default its size is 25% of its parent client area and it is positioned in the centre of its parent object.

The [Section 6.452](#) property specifies the location of the **internal** top-left corner of the SubForm relative to its parent. If the SubForm has a title bar, border, or a 3-dimensional shadow, you must allow sufficient space for these components. Similarly, the [Section 6.556](#) property specifies the internal size of the SubForm excluding the title bar and border.

A SubForm is constrained so that it cannot be moved outside its parent. In all other respects it behaves in a similar manner to a [Section 6.227](#) object. See [Section 6.227](#) object and the descriptions of its properties for further details.

6.579 SysColorChange Event 134

Applies To: [Section 6.489](#)

Description

If enabled, this event is reported when the user or another application updates the system colour palette. The event is reported after the change has taken place and cannot be disabled or inhibited in any way. If you want your application to respond to colour palette changes, this event gives you the opportunity of doing so.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'SysCoLoRChange' or 134

6.580 SysMenu Property

Applies To: [Section 6.227](#), [Section 6.307](#), [Section 6.578](#)

Description

This property determines whether or not a [Section 6.227](#) or [Section 6.578](#) has a "System Menu " box in the top-left corner of its border. Pressing the left mouse button in this box brings up the standard window control menu for the [Section 6.227](#). Double clicking the box closes the [Section 6.227](#).

SysMenu is a single number with the value 0 (no System menu box) or 1 (System Menu box is provided). The default is 1.

If any of the SysMenu, [Section 6.369](#), [Section 6.388](#) and [Section 6.398](#) properties are set to 1, the [Section 6.227](#) or [Section 6.578](#) has a title bar.

6.581 SysTrayItem Object

Purpose: The SysTrayItem object represents an item that you can create in the Windows System Tray.

Parents Form Root

Children Icon Menu Timer

Properties Type Event IconObj Data Tip Translate Popup KeepOnClose MethodList
ChildList EventList PropList

Methods Detach ShowBalloonTip Wait

Events Close Create MouseDown MouseUp MouseMove MouseDbClick BalloonShow
BalloonHide BalloonTimeout BalloonUserClick

Description

The SysTrayItem object appears as an icon in the Windows System Tray and allows the user to interact with your application even if it is minimised or has no other visible presence.

Interaction is provided through a pop-up menu that is displayed when the user clicks on the SysTrayItem. The SysTrayItem does not support mouse or keyboard events directly.

The IconObj property specifies the name of an Icon object used to display the SysTrayItem. If not specified, the default is the standard Dyalog APL icon.

The Popup property specifies the name of a Menu object (which may be a child of the SysTrayItem). The Menu object is displayed automatically when the user clicks on the SysTrayItem icon. The Menu should contain one or more MenuItem objects with suitable callback functions attached.

Unlike other popup menus, the SysTrayItem menu is not activated by an explicit (modal) ⌘ but is posted automatically for you. The MenuItem callbacks will be executed by the current ⌘, with the exception of modal ⌘s on MsgBox, FileBox, Locator and other popup Menu objects. For example, if your application is in a modal ⌘ on a Form, that ⌘ will react to and action events on the SysTrayItem menu, even though it is not explicitly included in the list of objects being ⌘'ed.

The Tip property specifies a character string to be displayed when the user hovers the mouse over the SysTrayItem. This is displayed using the user's current setting for Tip text and it is not possible to change this appearance.

6.582 TabBar Object

Purpose: To manage a set of [Section 6.583](#) objects.

Parents ActiveXControl Form SubForm

Children Circle Ellipse Font Marker Poly Rect TabBtn Text Timer

Properties Type Posn Size Coord Align Active Visible Event VScroll HScroll Sizeable FontObj BCol Picture OnTop IconObj CursorObj AutoConf YRange XRange Data Attach TextSize Handle Hint HintObj Tip TipObj TabObj Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel Select

Description

The TabBar object manages a group of [Section 6.583](#) objects. These are associated with a set of [Section 6.578](#) objects which are positioned on top of one another. When the user clicks on a [Section 6.583](#), the corresponding [Section 6.578](#) is brought to the top and given the focus.

TabBar and [Section 6.583](#) objects were implemented before Windows provided direct support for tabbed dialogs, and have been superseded by [Section 6.585](#) and [Section 6.584](#) objects. Please use these instead.

By default, a TabBar is a flat bar stretched across the bottom of its parent form. You can alter its appearance using its [Section 6.184](#) property and you can control its alignment with its [Section 6.13](#) property. [Section 6.13](#) can be set to Top, Bottom (the default), Left or Right and causes the TabBar to be attached to the corresponding edge of the [Section 6.227](#). A TabBar aligned Top or Bottom will automatically stretch or shrink horizontally when its parent [Section 6.227](#) is resized, but it will remain fixed vertically. A TabBar aligned Left or Right will stretch vertically but will remain fixed horizontally. By default a TabBar occupies the entire width or length of the side of the [Section 6.227](#) to which it is attached. Both the [Section 6.452](#) and [Section 6.556](#) properties can be altered.

The alignment of a TabBar also determines the orientation of its [Section 6.583s](#). TabBars aligned Top or Bottom cause their [Section 6.583s](#) to be drawn left to right with the free edge of the [Section 6.583s](#) facing downwards or upwards respectively. TabBar aligned Left or Right draw their [Section 6.583s](#) downwards with their free edges facing left or right respectively.

By default, [Section 6.583](#) objects are positioned along the inner edge of the TabBar. This is the edge closest to the [Section 6.578](#) s they will tab. They are also positioned so that they overlap one another horizontally or vertically according to the [Section 6.13](#) property.

The [Section 6.303](#) and [Section 6.646](#) properties determine what happens when the end of the TabBar is reached. If [Section 6.303](#) or [Section 6.646](#) is 0 (the default) a [Section 6.583](#) that would otherwise extend beyond the TabBar is instead positioned immediately above, below or alongside the first [Section 6.583](#) in the TabBar, thereby starting a new row or column. Note however that the TabBar is not automatically resized vertically to accommodate a second row or column. If you want a multi-flight TabBar you have to set its height or width explicitly. If [Section 6.303](#) or [Section 6.646](#) is -1 or -2, [Section 6.583](#)s continue to be added along the TabBar even though they extend beyond its boundary and may be scrolled into view using a mini scrollbar. If [Section 6.303](#) is -1, the scrollbar is shown whether or not any controls extend beyond the TabBar. If [Section 6.303](#) is -2, the scrollbar appears only if required and may appear or disappear when the user resizes the parent [Section 6.227](#).

[Section 6.646](#) and [Section 6.303](#) may only be set when the object is created and may not subsequently be changed.

If you specify a value for its [Section 6.452](#) property, a [Section 6.583](#) will be placed at the requested position regardless of the value of [Section 6.577](#), [Section 6.303](#) or [Section 6.646](#). However, the next control added will take its default position from the previous one according to the value of these properties. Thus if you wish to group your controls together with spaces between the groups, you need only specify the position of the first one in each group.

If you specify a value for its [Section 6.452](#) property, a [Section 6.583](#) will be placed at the requested position regardless of the value of [Section 6.13](#). However, the next [Section 6.583](#) added will take its default position from the previous one. Thus if you wish to group your [Section 6.583](#)s together with spaces between the groups, you need only specify the position of the first one in each group.

6.583 TabBtn Object

Purpose: To tab a [Section 6.578](#) .

Parents TabBar

Children Timer

Properties Type Caption Posn Size Align Border Active Visible Event FontObj FCol BCol AutoConf Data Attach EdgeStyle TabObj Translate Accelerator KeepOnClose MethodList ChildList EventList PropList

Methods Detach ChooseFont

Events Close Create FontOK FontCancel DropObjects Select MouseDown MouseUp
MouseMove MouseDbClick

Description

TabBtn objects are associated with [Section 6.578](#)s which are positioned on top of one another. When the user clicks on a TabBtn, the corresponding [Section 6.578](#) is brought to the top and given the focus.

[Section 6.582](#) and TabBtn objects were implemented before Windows provided direct support for tabbed dialogs, and have been superseded by [Section 6.585](#) and [Section 6.584](#) objects. Please use these instead.

The appearance of a TabBtn is determined by its [Section 6.184](#), [Section 6.49](#) and [Section 6.66](#) properties. These take their defaults from the [Section 6.578](#) with which the TabBtn is associated. Thus there is generally no need to specify them. [Section 6.45](#) also defaults to that of its associated [Section 6.578](#).

The position of a TabBtn is normally determined by its parent [Section 6.582](#) and its default size is fixed (22 x 80 pixels), and not related to the size of its [Section 6.66](#). These defaults can be overridden using the [Section 6.452](#) and [Section 6.556](#) properties.

A [Section 6.578](#) is associated with a TabBtn by setting the [Section 6.589](#) property of the [Section 6.578](#) to the name of, or ref to, the TabBtn. The [Section 6.589](#) property of the TabBtn is a read-only property that contains the name of, or ref to, the associated [Section 6.578](#).

6.584 TabButton Object

Purpose: The TabButton object represents an individual tab or button in a [Section 6.585](#)

Parents TabControl

Children Timer

Properties Type Caption Posn Size State Event ImageIndex Data Tip TabObj Accelerator
KeepOnClose MethodList ChildList EventList PropList

Methods Detach

Events Close Create Select

Description

The TabButton object represents an individual tab or button in a [Section 6.585](#)

The position and size of a TabButton object are entirely determined by its parent [Section 6.585](#) and may not be altered. For this reason, the [Section 6.452](#) and [Section 6.556](#) properties are read-only.

The [Section 6.66](#) property specifies the text that appears on the button or tab.

A picture is specified by setting the ImageIndex property of the TabButton. This is a number that points to a particular icon or bitmap defined in an [Section 6.316](#) object whose name is specified by the [Section 6.317](#) property of the parent [Section 6.585](#).

Note that all TabButton objects share the same font which is defined by the [Section 6.225](#) property of the [Section 6.585](#).

The foreground and background colours of the TabButton object are fixed.

When used as a tab, a TabButton is normally attached to a [Section 6.578](#) by the [Section 6.589](#) property of the [Section 6.578](#). The [Section 6.589](#) property of the TabButton itself, is a read-only property that specifies the name of, or ref to, the [Section 6.578](#) to which the TabButton is attached.

The [Section 6.571](#) property reports the (selected) state of a TabButton.

6.585 TabControl Object

Purpose: The TabControl object provides access to the native Windows tab control.

Parents ActiveXControl CoolBand Form SubForm

Children ImageList SubForm TabButton Timer

Properties Type Posn Size Style Visible Event ImageListObj FontObj Data Attach Handle TabObj KeepOnClose MultiLine TabSize Justify TabJustify Align MultiSelect TabFocus HotTrack ScrollOpposite FlatSeparators MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create

Description

The standard tab control is analogous to a set of dividers in a notebook and allows you to define a set of *pages* that occupy the same area of a window or dialog box. 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. For example, the Windows taskbar is such a tab control.

The overall appearance of the TabControl is determined by the [Section 6.577](#) property which may be 'Tabs' (the default), 'Buttons' or 'FlatButtonButtons'.

Individual tabs or buttons are represented by [Section 6.584](#) objects which should be created as children of the TabControl object. Optional captions and pictures are specified by the [Section 6.66](#) and [Section 6.315](#) properties of the individual [Section 6.584](#) objects themselves. Otherwise, the appearance of the tabs or buttons is determined by properties of the TabControl itself.

To implement a multiple page tabbed dialog [Section 7.11](#), you should create a [Section 6.227](#), then a TabControl with Style 'Tabs' as a child of the [Section 6.227](#). Next, create one or more pairs of [Section 6.584](#) and [Section 6.578](#) objects as children of the TabControl. You associate each [Section 6.578](#) with a particular tab by setting its [Section 6.589](#) property to the name of, or ref to, the associated [Section 6.584](#) object. Making the [Section 6.578](#) children of the TabControl ensures that, by default, they will automatically be resized correctly. You may alternatively create your [Section 6.578](#) as children of the main [Section 6.227](#) and establish appropriate resize behaviour using their [Section 6.32](#) property.

A TabControl object with [Section 6.577](#) 'Buttons' [Section 7.12](#) or 'FlatButtonButtons' [Section 7.13](#) may be used in a similar way (that is, to display a set of alternative pages), although buttons in this type of TabControl are more normally used to execute commands. For this reason, these styles of TabControl are without borders.

If [Section 6.577](#) is 'FlatButtonButtons', the [Section 6.220](#) property specifies whether or not separators are drawn between the buttons. The default value of [Section 6.220](#) is 0 (no separators).

[Section 7.14](#) shows the effect of setting [Section 6.220](#) to 1.

The [Section 6.13](#) property specifies along which of the 4 edges of the TabControl the tabs or buttons are arranged. [Section 6.13](#) also controls the relative positioning of the picture and Caption within each TabButton. [Section 6.13](#) may be Top (the default), Bottom, Left or Right.

If [Section 6.13](#) is 'Top' or 'Bottom' [Section 7.15](#), the tabs or buttons are arranged along the top or bottom edge of the TabControl and picture is drawn to the left of the Caption.

If [Section 6.13](#) is 'Left' [Section 7.16](#), the tabs or buttons are arranged top-to-bottom along the left edge of the TabControl, and the pictures are drawn below the Captions.

[Section 6.13](#) 'Align' 'Right' fails to show the tabs or buttons on the right hand edge of the TabControl; this appears to be a limitation of Windows.

The [Section 6.32](#) property specifies how the TabControl responds when its parent is resized. Its default value, which is independent of the [Section 6.13](#) property, is 'None' 'None' 'None' 'None'. This causes the TabControl to maintain its original proportions when its parent is resized.

The [Section 6.404](#) property determines whether or not your tabs or buttons will be arranged in multiple flights or multiple rows/columns.

The default value of [Section 6.404](#) is 0, in which case, if you have more tabs or buttons than will fit in the space provided, the TabControl displays an UpDown control to permit the user to scroll them. [Section 7.18](#)

If [Section 6.404](#) is set to 1, the tabs are displayed in multiple flights [Section 7.19](#) or the buttons are displayed in multiple rows [Section 7.20](#).

The [Section 6.510](#) property specifies that unneeded tabs scroll to the opposite side of a TabControl, when a tab is selected. Setting [Section 6.510](#) to 1 forces [Section 6.404](#) to 1 also.

[Section 7.21](#) illustrates a TabControl with [Section 6.510](#) set to 1, after the user has clicked *Third Tab*. Notice that, in this example, the SubForms have been created as children of the TabControl. This is necessary to ensure that they are managed correctly in this case.

If [Section 6.404](#) is 1, the way that multiple flights of tabs or rows/columns of buttons are displayed is further defined by the Justify property which may be 'Right' (the default) or 'None'.

If [Section 6.336](#) is 'Right' (which is the default), the TabControl increases the width of each tab, if necessary, so that each row of tabs fills the entire width of the tab control. Otherwise, if [Section 6.336](#) is empty or 'None', the rows are ragged.

See [Section 7.22](#) (for tabs) and [Section 7.23](#) (for buttons).

By default, the size of the tabs may vary from one to another. Fixed size tabs may be obtained by setting the [Section 6.590](#) property.

To obtain fixed sized tabs with [Section 6.404](#) set to 1, you must however also set Justify to 'None'.

If fixed size tabs are in effect, the positions at which the picture and Caption are drawn within each [Section 6.584](#) is controlled by the [Section 6.588](#) property which may be 'Centre', 'Edge', or 'IconEdge'.

[Section 7.24](#) illustrates these different settings.

The font used to draw the captions in the [Section 6.584](#) objects is determined by the [Section 6.225](#) property of the TabControl.

You cannot specify the foreground or background colours of the tabs/buttons, nor can you use different fonts in different tabs/buttons. The orientation of the Caption text is always determined by the value of the Align property of the TabControl.

The [Section 6.589](#) property is read-only and reports the name of, or ref to, the [Section 6.584](#) that is currently selected.

The [Section 6.405](#) property specifies whether or not the user can select more than one button in a TabControl at the same time, by holding down the Ctrl key when clicking. The default is 0 (only one button may be selected). [Section 6.405](#) is ignored if Style is 'Tabs'.

The [Section 6.586](#) property specifies the focus behaviour for the TabControl object and may be 'Normal' (the default), 'Never' or 'ButtonDown'.

The [Section 6.302](#) property specifies whether or not the tabs or buttons are automatically highlighted by the mouse pointer. The default is 0 (no highlighting).

6.586 TabFocus Property

Applies To: [Section 6.585](#)

Description

The TabFocus property specifies the focus behaviour for the [Section 6.585](#) object and may

TabFocus is a character vector that may be 'Normal' (the default), 'Never' or 'ButtonDown'.

If TabFocus is 'Normal', the tabs or buttons in a [Section 6.585](#) do not immediately receive the input focus when clicked, but only when clicked a second time. This means

that, normally, when the user circulates through the tabs, the input focus will be given to the appropriate control in the associated SubForm. However, if the user clicks twice in succession on the same tab or button, the [Section 6.585](#) itself will receive the input focus.

If TabFocus is 'ButtonDown', the tabs or buttons in a [Section 6.585](#) receive the input focus when clicked.

If TabFocus is 'Never', the tabs or buttons in a [Section 6.585](#) never receive the input focus. This allows the user to circulate through a set of tabbed SubForms without ever losing the input focus to the [Section 6.585](#) itself.

6.587 TabIndex Property

Applies To

Description

The TabIndex property reports the $\square IO$ -dependent relative position of a child object within the list of child objects owned by its parent. If N is the number of children owned by an object, TabIndex is an integer between $\square IO$ and $(N - \sim \square IO)$. The sequence of objects in this list is also used as the tabbing sequence, that is, if the input focus is on the first child in the list, pressing Tab moves the input focus to the next child in the list.

When you create a child object, it is inserted in the list at the position specified by its TabIndex property. If TabIndex is omitted, it is appended to the end of the list.

If you subsequently change TabIndex, the object is moved to the corresponding position in the list.

Naturally, if you specify a value of TabIndex that is greater than the number of existing children, the object is inserted at or moved to the end of the list.

6.588 TabJustify Property

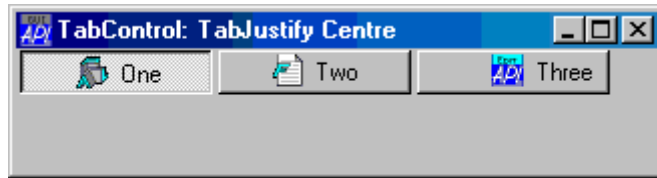
Applies To: [Section 6.585](#)

Description

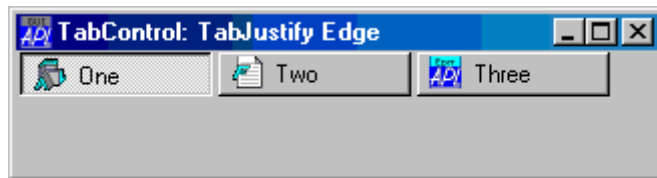
The TabJustify property specifies, the positions at which the picture and caption are drawn within each tab or button implemented by a [Section 6.584](#) in a TabControl object.

TabJustify is a character vector that may be 'Centre', 'Edge', or 'IconEdge'. Its default value is 'Centre'.

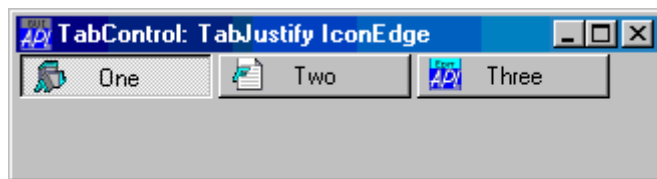
If TabJustify is 'Centre', the picture and caption are arranged in the centre of the [Section 6.584](#).



If TabJustify is 'Edge', the picture and caption are together aligned to the appropriate edge of the [Section 6.584](#) according to the value of Align.



If TabJustify is set to 'IconEdge', the caption is drawn centrally and only picture is aligned to the edge.



TabJustify is only honoured if fixed size tabs are in effect.

6.589 TabObj Property

Applies To: [Section 6.578](#), [Section 6.582](#), [Section 6.583](#), [Section 6.584](#), [Section 6.585](#)

Description

TabObj is a ref or a character vector.

TabObj associates a [Section 6.578](#) with a [Section 6.583](#) or a [Section 6.584](#) object. Selecting the associated [Section 6.583](#) or [Section 6.584](#) causes the [Section 6.578](#) to be given the input focus.

For a [Section 6.578](#), it specifies the name of, or ref to, a [Section 6.583](#) or [Section 6.584](#) object that is to be associated with the [Section 6.578](#). When referenced or queried using `WG`, `TabObj` returns a name if it was specified by a name, or a ref if it was specified by a ref.

For [Section 6.583](#) and [Section 6.584](#) objects, `TabObj` is a read-only property that contains a ref to the associated [Section 6.578](#).

For a [Section 6.582](#) or [Section 6.585](#), `TabObj` is a read-only property that contains a ref to the currently selected [Section 6.583](#) or [Section 6.584](#).

6.590 TabSize Property

Applies To: [Section 6.585](#)

Description

The `TabSize` property specifies the size of fixed size tabs or buttons in a `TabControl` object.

By default, the size of the tabs may vary from one to another. Fixed size tabs may be obtained by setting the `TabSize` property.

`TabSize` is a 2-element numeric vector that specifies the height and width of the tab. The first element of `TabSize` may be set to 0 which means "default height".

To obtain fixed sized tabs with [Section 6.404](#) set to 1, you must however also set the `Justify` property to 'None'.

If [Section 6.404](#) is 1 and `Justify` is 'Right', `TabSize` is ignored.

6.591 Target Property

Applies To: [Section 6.51](#)

Description

The `Target` property is a read-only character string that specifies the chosen folder or other resource selected by the user in a [Section 6.51](#) object.

If the [Section 6.52](#) is 'Directory', `Target` will contain a directory name followed by the character "\". Otherwise, `Target` just contains the name.

6.592 TargetState Property

Applies To: [Section 6.604](#)

Description

The TargetState property reflects the intended final state of a [Section 6.604](#) object. Its possible values are as follows:

Stream	UDP
Client	Open
Server	Bound
Closed	Closed

Setting TargetState to Closed is the recommended way to close a socket. It informs APL that you want the socket to be closed, but only when it is safe to do so. When all the data has been sent, the [Section 6.604](#) will generate a [Section 6.594](#) event which, unless a callback function decides otherwise, will cause the [Section 6.604](#) object to disappear.

To control socket closure, you may execute the following steps:

1. Set TargetState to Closed
2. **Either:**
 1. continue processing **or**
 2. wait (using `DDQ`) for the [Section 6.604](#) to disappear **or**
 3. wait (using `DDQ`) for the [Section 6.594](#) event

6.593 TCPAccept Event 371

Applies To: [Section 6.604](#)

Description

If enabled, this event is reported when a client connects to a server [Section 6.604](#) object.

You may not disable or nullify the operation by setting the action code for the event to `-1` or by returning 0 from a callback function. You may also not call TCPAccept as a method or generate this event artificially using `INQ`.

The event message reported as the result of `FDQ`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'TCPAccept ' or 371
[3]	Socket handle	an integer

The socket handle reported by this event is the socket handle for the original listening socket that was associated with the [Section 6.604](#) before the client connected.

If you want your server to remain available for other clients, you must create a new [Section 6.604](#) object in a callback function attached to this event. The new [Section 6.604](#) object must be created by cloning the original listening socket. This is done by specifying the socket handle as the value of its `SocketNumber` property. You may not specify any other properties (except `Event` and `Data`) in the `WC` statement that creates the new clone object.

The default processing for this event is to close the socket handle reported by the 3rd element of the event message **unless** it has been associated with a new [Section 6.604](#) object by the callback function as described above. You may prevent this from occurring by returning 0 from a callback function. This may be necessary in a multithreaded application.

You may not call `TCPClose` as a method or generate this event artificially using `FNQ`.

6.594 TCPClose Event 374

Applies To: [Section 6.604](#)

Description

If enabled, this event is reported when the remote end of a TCP/IP connection breaks the connection.

You may not disable or nullify the operation by setting the action code for the event to `-1` or by returning 0 from a callback function. You may also not call `TCPClose` as a method or generate this event artificially using `FNQ`.

The event message reported as the result of `FDQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' TCPClose ' or 374

6.595 TCPConnect Event 372

Applies To: [Section 6.604](#)

Description

If enabled, this event is reported when a server accepts the connection of a client [Section 6.604](#) object and is reported by the client.

You may not disable or nullify the operation by setting the action code for the event to -1 or by returning 0 from a callback function. You may also not call TCPConnect as a method or generate this event artificially using `INQ`.

The event message reported as the result of `INQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' TCPConnect ' or 372

6.596 TCPError Event 370

Applies To: [Section 6.604](#)

Description

This event is generated when a fatal TCP/IP error occurs and is reported by a [Section 6.604](#) object.

The default processing for this event is to display a message box containing details of the TCP/IP error. You may disable the display of this message box by setting the action code for the event to -1 or by returning 0 from a callback function attached to it.

The event message reported as the result of `INQ`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'TCPError' or 370
[3]	Error code	a number
[4]	Error text	a character vector

6.597 TCPGetHostID Method 376

Applies To: [Section 6.489](#), [Section 6.604](#)

Description

This method is used to obtain the IP Address of your PC.

The TCPGetHostID method is niladic.

The (shy) result is a character string containing your IP address. If you have more than one, it will return the first.

For example:

```
TCPGetHostID
193.32.236.43
```

6.598 TCPGotAddr Event 377

Applies To: [Section 6.604](#)

Description

If enabled, this event is reported when a host name (specified by the [Section 6.475](#) or [Section 6.351](#) property) is resolved to an IP address.

You may not disable or nullify the operation by setting the action code for the event to -1 or by returning 0 from a callback function. You may also not call TCPGotAddr as a method or generate this event artificially using `INQ`.

The event message reported as the result of `INQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' TCPGotAddr ' or 377

Note that the IP address is not reported in the event message but may be obtained from [Section 6.474](#) or [Section 6.350](#) as appropriate.

6.599 TCPGotPort Event 378

Applies To: [Section 6.604](#)

Description

If enabled, this event is reported when a port name (specified by the [Section 6.477](#) or [Section 6.354](#) property) is resolved to a port number.

You may not disable or nullify the operation by setting the action code for the event to -1 or by returning 0 from a callback function. You may also not call TCPGotPort as a method or generate this event artificially using `□NQ`.

The event message reported as the result of `□DQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	' TCPGotPort ' or 378

Note that the port number is not reported in the event message but may be obtained from [Section 6.476](#) or [Section 6.353](#) as appropriate.

6.600 TCPReady Event 379

Applies To: [Section 6.604](#)

Description

If enabled, this event is reported when the TCP/IP buffers are free and there is no data waiting to be sent in the internal APL queue.

This event is provided to enable you to control the transmission of a large amount of data that cannot be handled in a single call to [Section 6.602](#).

The amount of data that the system can handle in one go is limited by TCP/IP buffers, the speed of the network, and the amount of Windows memory and disk space available for buffering.

You may not disable or nullify the operation by setting the action code for the event to `-1` or by returning 0 from a callback function. However, you *may* call `TCPReady` as a method or generate this event artificially using `INQ`.

The event message reported as the result of `INQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'TCPReady' or 379

6.601 TCPRecv Event 373

Applies To: [Section 6.604](#)

Description

If enabled, this event is reported when data is received by a [Section 6.604](#) object.

You may not disable or nullify the operation by setting the action code for the event to `-1` or by returning 0 from a callback function. You may also not call `TCPRecv` as a method or generate this event artificially using `INQ`.

The event message reported as the result of `INQ`, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'TCPRecv' or 373
[3]	Data	the data received
[4]	IP address	character vector
[5]	Port number	integer

Elements [4-5] refer to the IP address and port number of the remote process that sent the data.

If the [Section 6.560](#) is 'Stream', this information will be identical to the values of the [Section 6.474](#) and [Section 6.476](#) respectively.

If the [Section 6.560](#) is 'UDP' and there is potentially more than one partner sending you data, the IP address and port number information provided by the TCPRecv event is more reliable than the current values of [Section 6.474](#) and [Section 6.476](#) as these may already have changed.

6.602 TCPSend Method 375

Applies To: [Section 6.604](#)

Description

This method is used to send data to a remote process connected to a [Section 6.604](#) object

The argument to TCPSend is a 1 or 3-element array as follows:

[1]	Data	the data to be sent
[2]	IP address	character vector
[3]	Port number	integer

If Style is 'Char', the data to be sent must be a character vector. If Style is 'Raw', the data to be sent must be an integer vector whose elements are in the range -128 to 255. If Style is 'APL', any array may be transmitted.

The optional *IP address* and *Port number* parameters specify the intended recipient of the message and apply only if the [Section 6.560](#) is 'UDP', in which case they are mandatory. If the [Section 6.560](#) is 'Stream', these parameters will be ignored and should be omitted.

6.603 TCPSendPicture Method 380

Applies To: [Section 6.604](#)

Description

This method is used to transmit a picture represented by a [Section 6.47](#) object to a TCP/IP socket. The picture may be transmitted in GIF or in PNG format.

The argument to TCPSendPicture is a 1 or 2-element array as follows:

[1]	Bitmap name	character vector
[2]	Picture format	character vector, 'GIF ' or 'PNG '

If *Picture format* is omitted, the default is GIF format.

Note that the [Section 6.577](#) of the [Section 6.604](#) object must be set to 'Raw' before you execute the TCPSendPicture method.

The (shy) result of the method is an integer that reports the number of bytes that were transmitted.

EXAMPLE

```
S1.TCPSendPicture 'BM' 'PNG'
4930
```

Note

Although PNG is recognised as the latest graphics standard for displaying pictures, not all Web browsers support it.

See also: [Section 6.362](#), [Section 6.363](#)

6.604 TCPSocket Object

Purpose: The TCPSocket object provides an interface to TCP/IP.

Parents ActiveXControl Calendar CoolBand DateTimePicker Form NetType OLEClient OLEServer PropertyPage Root SubForm TCPSocket

Children Bitmap BrowseBox Clipboard Cursor FileBox Font Form Icon ImageList Locator Menu Metafile MsgBox OCXClass OLEClient Printer PropertySheet TCPSocket Timer TipField

Properties Type LocalAddr LocalPort RemoteAddr RemotePort Style Event LocalAddrName LocalPortName RemoteAddrName RemotePortName Data SocketType SocketNumber CurrentState TargetState KeepOnClose Encoding MethodList ChildList EventList PropList

Methods Detach TCPSend TCPGetHostID TCPSendPicture Wait

Events Close Create TCPError TCPAccept TCPConnect TCPRecv TCPClose TCPGotAddr TCPGotPort TCPReady

Description

The TCPsocket object provides an event-driven mechanism to communicate with other programs (including Dyalog APL) via TCP sockets. Dyalog recommends that Conga is used in preference to TCPSockets in new applications.

The [Section 6.560](#) property is a character vector that specifies the type of the TCP/IP socket. This is either 'Stream' (the default), or 'UDP'. [Section 6.560](#) must be defined when the object is created and cannot be set or changed using `⎕WS`.

The Style property is a character vector that specifies the type of data transmitted or received by the socket; it may be 'Char', 'Raw', or 'APL'. The value 'APL' is valid only if the [Section 6.560](#) is 'Stream'.

The [Section 6.191](#) property is a character vector that specifies how character data are encoded or translated. The possible values are 'None', 'UTF-8', 'Classic' or 'Unicode', depending upon the value of the Style property.

[Section 6.350](#) and [Section 6.353](#) properties identify your end of the connection; [Section 6.474](#) and [Section 6.476](#) identify the other end of the connection. The values of the two sets of properties are clearly symmetrical; your [Section 6.350](#) is your partner's [Section 6.474](#), and there are strict rules concerning which of them you and your partner may set. See the individual descriptions of these properties for details.

The [Section 6.559](#) property is the Window handle of the socket attached to the TCPsocket object and is generally a read-only property. The only time that [Section 6.559](#) may be specified is when a server replicates (clones) a listening socket to which a client has just connected.

6.605 Text Object

Purpose: Writes text.

Parents ActiveXControl Animation Bitmap Button ButtonEdit Combo ComboEx Edit Form Grid Group Label List ListView MDIClient Metafile Printer ProgressBar PropertyPage PropertySheet RichEdit Scroll Spinner Static StatusBar SubForm TabBar TipField ToolBar TrackBar TreeView UpDown

Children Timer

Properties Type Text Points FCol BCol VAlign HAlign Coord Active Visible Event Draggable
FontObj OnTop CursorObj AutoConf Data Translate Accelerator KeepOnClose
DrawMode MethodList ChildList EventList PropList

Methods Detach ChooseFont

Events Close Create FontOK FontCancel DragDrop MouseDown MouseUp MouseMove
MouseDown Click Help Select

Description

The Text object is used to write arbitrary text. It can be used in a [Section 6.227](#), [Section 6.578](#) or [Section 6.279](#) instead of a [Section 6.341](#). The main difference is that a [Section 6.341](#) is implemented as a true window object (thus consuming Windows resources). A Text object is not a window and consumes no MS-Windows resources. However, a [Section 6.341](#) supports [Section 6.174](#) events and has various useful properties that are not shared by the Text object.

The contents of the Text object are defined by its [Section 6.606](#) property. This is a character array containing one of the following :

- a simple scalar
- an enclosed vector or matrix (also a scalar)
- a simple vector
- a simple matrix
- a vector of enclosed vectors or matrices

[Section 6.449](#) is either a simple 2-column matrix of (y,x) co-ordinates, or a 2-element vector of y-coordinates and x-coordinates respectively.

There are two distinct cases:

1. [Section 6.449](#) specifies a single point. In this case, Text may be a single scalar character, a simple vector, or a matrix containing a block of text. The result is that the character, string, or matrix is written at the specified point.
2. [Section 6.449](#) specifies more than one point. There are three possibilities:
 1. If Text is a scalar, its contents are written at each of the points in [Section 6.449](#). This means that by enclosing a vector or matrix, you can draw a string or block of text at several locations.
 2. If Text is a vector, each element of Text is written at the corresponding position in [Section 6.449](#).

3. If Text is a matrix, each row of Text is written at the corresponding position in [Section 6.449](#).

[Section 6.225](#) specifies a **single** font to be used to write the Text. See a description of the [Section 6.225](#) property for details.

[Section 6.205](#) specifies the colour of the Text. For a single text item, FCol may be a single number which specifies one of the standard MS-Windows colours, or a simple 3-element numeric vector of RGB colour intensities. If more than one text item is involved, [Section 6.205](#) may be a vector which specifies the colour for each item separately. If so, its length must be the same as the number of points specified by [Section 6.449](#).

[Section 6.45](#) specifies the background colour of the text, that is, the colour for the part of the character cell that is blank. It is defined in the same way as [Section 6.205](#).

[Section 6.280](#) and [Section 6.640](#) specify the horizontal and vertical alignment of the text respectively. They may each be numeric scalars or vectors with the same length as the number of points specified in [Section 6.449](#). See [Section 6.280](#) and [Section 6.640](#) for details.

When one or more of [Section 6.205](#), [Section 6.45](#), [Section 6.280](#) and [Section 6.640](#) are vectors, the different components of Text are drawn using the corresponding colours and alignments.

The value of the [Section 6.173](#) property specifies whether or not the Text object can be dragged by the user. The value of the [Section 6.35](#) property determines whether or not the Text object is repositioned when its parent is resized.

EXAMPLES

Write 'A' at (10,20)

```
'g.t1' WC 'Text' 'A' (10 20)
```

Write 'h' at (10,20) in red

```
'g.t1' WC 'Text' 'h' (10 20) ('FCol' 255 0 0)
```

Write 'Hello' at (10,20)

```
'g.t1' WC 'Text' 'Hello' (10 20)
```

```
Write 'THIS IS A
      BLOCK OF
      TEXT      ' at (20,30)
```

```
BLK←3 9p'THIS IS A BLOCK OF TEXT      '
'g.t1' □WC 'Text' BLK (10 20)
```

Write 'A' at (10,20) and at (30,40)

```
'g.t1' □WC 'Text' 'A' ((10 30)(20 40))
```

Write a red '+' at (10,20) and a green '+' at (20,40)

```
'g.t1' □WC 'Text' '+' ((10 30)(20 40))('FCol' (255 0 0)(0 255 0))
```

Write 'Hello' at (10,20) and at (30,40)

```
'g.t1' □WC 'Text' (<'Hello') ((10 30)(20 40))
```

Write 'A' at (10,20) and 'B' at (30,40)

```
'g.t1' □WC 'Text' 'AB' ((10 30)(20 40))
```

Write 'Hello' at (10,20) and 'World' at (30,40)

```
'g.t1' □WC 'Text' ('Hello' 'World')((10 30)(20 40))
```

6.606 Text Property

Applies To

Description

This property is associated with the text contents of an object and is a character array.

In a [Section 6.56](#), [Section 6.121](#), [Section 6.575](#), [Section 6.563](#), or a single-line [Section 6.185](#) object, Text may be a simple scalar or a simple vector.

In a [Section 6.488](#), a multi-line [Section 6.185](#) field or in a [Section 6.399](#), the value of Text may also be a simple matrix, or a vector of vectors. If so, "new-line" characters are appended to each row of the matrix, or to each vector in a vector of vectors, before being displayed. The user may insert or add a "new-line" character in a multi-line [Section 6.185](#) by pressing Ctrl-Enter (Enter itself is used to press [Section 6.55s](#)).

Note that if word-wrapping is in effect in a multi-line [Section 6.185](#) object, the structure of Text does not correspond to the lines displayed.

In a [Section 6.605](#) object, the value of the Text property may be a simple scalar, an enclosed vector or matrix, a simple vector, a simple matrix, or a vector of enclosed vectors or matrices.

In general, the value of Text returned by *Dyalog APL Language: Wg* has the same structure that was assigned to it by *Dyalog APL Language: Wc* or by the most recent call to *Dyalog APL Language: Ws*. New-Line characters entered by the users are removed.

You can copy text into the Windows [Section 6.99](#) by using *Dyalog APL Language: Ws* to set Text for a [Section 6.99](#) object. In this case you may specify a simple character scalar, vector or matrix, or a vector of character vectors. If you are retrieving data from the clipboard that has been stored by another application, Text will be either a character vector or a vector of character vectors.

The Text property of a [Section 6.575](#) is updated automatically if its [Section 6.577](#) property is set to monitor the status of a key.

6.607 TextSize Property

Applies To

Description

This property has been replaced by the [Section 6.256](#) method, which should be used instead. TextSize is retained only for compatibility with previous versions of Dyalog.

TextSize is a "read-only" property that reports the size of the bounding rectangle of a text item in a given font. The result is given in the co-ordinate system of the object in question. This property is useful for positioning [Section 6.605](#) objects.

When you query TextSize you give the text item in whose size you are interested and, optionally, the name of a [Section 6.222](#) object. The text item may be a simple scalar, a vector or a matrix. If the [Section 6.222](#) is omitted, the result is given using the current font for the object in question. When you query TextSize on its own, you must enclose the argument to *Dyalog APL Language: Wg*. This is because APL would otherwise not be able to distinguish between the text string and font name associated with 'TextSize' and other properties with the same name as these items.

EXAMPLES

```

    '.' □WG c'TextSize' 'Hello World'
2.666666746 9.625

    'FNT1' □WC 'FontObj' 'Arial' 72
    '.' □WG c'TextSize' 'Hello World' 'FNT1'
12 41.875

    '.' □WS 'Coord' 'Pixel'
    '.' □WG ('TextSize' (3 11p'Hello World')) 'Coord'
39 55 Pixel

```

6.608 Thumb Property

Applies To

Description

This property determines and reports the position of the *thumb* in an object.

For a [Section 6.508](#) object, the value of Thumb is a single integer whose minimum value is 1 and whose maximum value is defined by the [Section 6.469](#) property.

For [Section 6.459](#), [Section 6.563](#), [Section 6.637](#) and [Section 6.625](#) objects, Thumb is a single numeric value in the range specified by the [Section 6.345](#) property.

For a [Section 6.227](#) or [Section 6.578](#) object, Thumb is a 2-element vector whose elements refer to the position of the thumb in the object's own built-in vertical and horizontal scrollbars respectively.

For other objects, Thumb is a single numeric value in the range defined by the Limits property.

6.609 ThumbDrag Event 440

Applies To: [Section 6.508](#), [Section 6.625](#)

Description

If enabled, this event is generated when the user drags the thumb in a [Section 6.625](#) object. The event is reported *after* the value of the Thumb property has been updated

and is reported continuously as the thumb is dragged. You may not disable this event or alter its effect with a callback function.

The event message reported as the result of `ThumbDrag`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'ThumbDrag' or 440
[3]	Thumb value	Integer. The new value of the Thumb property resulting from the user dragging the thumb.

6.610 ThumbRect Property

Applies To: [Section 6.625](#)

Description

ThumbRect is a *read-only* property that reports the position and size of the bounding rectangle of the thumb in a [Section 6.625](#) object. It is a 4-element integer vector containing:

[1]	Vertical position of the top-left corner of the bounding rectangle
[2]	Horizontal position of the top-left corner of the bounding rectangle
[3]	Height of the bounding rectangle
[4]	Width of the bounding rectangle

6.611 TickAlign Property

Applies To: [Section 6.625](#)

Description

TickAlign determines the position of the tick marks in a [Section 6.625](#) object. For a horizontal [Section 6.625](#), TickAlign may be either 'Bottom' (the default), 'Top' or 'Both'. If TickAlign is 'Bottom', the ticks are drawn *below* the slider. If TickAlign is 'Top', the ticks are drawn above it. If TickAlign is 'Both', the ticks are drawn above and below.

For a vertical [Section 6.625](#), TickAlign may be either 'Right' (the default), 'Left', or 'Both' and similarly specifies to which side of the slider bar the ticks are drawn. Note that TickAlign may only be set when the TrackBar is created with `WC` and may not subsequently be altered using `WS`.

Note that ticks are not drawn if the value of [Section 6.289](#) is 0

6.612 TickSpacing Property

Applies To: [Section 6.625](#)

Description

The TickSpacing property specifies the spacing between each tick mark in a [Section 6.625](#) object. It is an integer between 1 and the maximum value of the [Section 6.625](#) which is defined by the 2nd element of the [Section 6.345](#) property.

For example, if you set ('Limits' 10 50) and you specify ('TickSpacing' 10) you will obtain 5 ticks corresponding to the values 10, 20, 30, 40 and 50 along the slider bar.

6.613 Timer Object

Purpose: To generate an action at regular intervals.

Parents ActiveXControl Animation Bitmap BrowseBox Button ButtonEdit Calendar Circle Clipboard Combo ComboEx CoolBand CoolBar Cursor DateTimePicker Edit Ellipse FileBox Font Form Grid Group HTMLRenderer Icon Image ImageList Label List ListView Locator Marker MDIClient Menu MenuBar MenuItem Metafile MsgBox NetClient NetType OLEClient OLEServer Poly Printer ProgressBar PropertyPage PropertySheet Rect RichEdit Root Scroll Separator SM Spinner Splitter Static StatusBar StatusField SubForm SysTrayItem TabBar TabBtn TabButton TabControl TCPSocket Text Timer TipField ToolBar ToolButton ToolControl TrackBar TreeView UpDown

Properties Type Interval Active Event Data KeepOnClose FireOnce MethodList ChildList EventList PropList

Methods Detach Wait

Events Close Create Timer

Description

The Timer object is used to generate an event at regular intervals. It can be used to produce animation and to implement "repeaters" such as spin buttons.

The [Section 6.327](#) property specifies how often the Timer generates events and is defined in milliseconds. Its default value is 1000.

The [Section 6.5](#) property determines whether or not the Timer generates events and can be used to switch the Timer off and on as required.

The [Section 6.216](#) property may be used to implement a once-off Timer event and has the value 0, 1 or 2.

Note that if you create a Timer object whose [Section 6.614](#) event generates an error (for example by attaching it to a non-existent callback) it may be very difficult or even impossible to type into the Session, because the error will be displayed over and over again. Care is therefore recommended.

6.614 Timer Event 140

Applies To: [Section 6.613](#)

Description

This event is generated at regular intervals by a [Section 6.613](#) object and is typically used to fire a callback function to perform a task repeatedly. Returning a 0 from a callback function attached to a Timer event has no effect. The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2 element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'Timer' or 140

6.615 Tip Property

Applies To

Description

The Tip property is a character vector or character matrix that specifies a "help" message which is to be displayed when the user positions the mouse pointer over the object. The Tip is displayed in a pop-up [Section 6.616](#) object specified by the [Section 6.617](#) property.

6.616 TipField Object

Purpose: To display pop-up help.

Parents ActiveXControl CoolBand Form Group OLEServer PropertyPage PropertySheet
Root SubForm TCPSocket

Children Circle Ellipse Font Marker Poly Rect Text Timer

Properties Type Event FontObj FCol BCol Data Translate KeepOnClose MethodList
ChildList EventList PropList

Methods Detach ChooseFont

Events Close Create FontOK FontCancel

Description

The TipField is used to display pop-up help when the user moves the mouse pointer over an object.

Most of the GUI objects supported by Dyalog APL/W have a [Section 6.615](#) and a [Section 6.617](#) property. [Section 6.617](#) specifies the name of, or ref to, a TipField object, and [Section 6.615](#) specifies a "help" message. The TipField automatically pops-up to display the [Section 6.615](#) when the user moves the mouse pointer over the object. It disappears when the user moves the mouse pointer away.

The TipField is a simple box with a 1-pixel black border in which the text specified by [Section 6.615](#) is displayed. [Section 6.205](#), [Section 6.45](#) and [Section 6.225](#) can be used to customise the appearance of the text within the box. [Section 6.205](#) specifies the colour of the text; [Section 6.45](#) specifies the background colour with which the box is filled. The default is black on yellow.

If you wish to display [Section 6.615s](#) for particular objects in different fonts and colours, you must create a separate TipField for each combination of colour and font you need.

6.617 TipObj Property

Applies To

Description

The TipObj property is a character vector or ref that specifies the name of, or ref to, a [Section 6.616](#) object in which the "help" message defined by the [Section 6.615](#) property is to be displayed. This message is displayed when the user positions the mouse pointer over the object.

Note that if TipObj is empty, its value is inherited from its parent. Thus setting TipObj on a [Section 6.227](#) defines the default [Section 6.616](#) (and thus the default appearance of all [Section 6.615s](#)) for all the controls in that [Section 6.227](#). Setting TipObj on [Section 6.489](#) defines the default [Section 6.616](#) for the entire application.

6.618 TitleHeight Property

Applies To: [Section 6.263](#)

Description

This property is a single number that specifies the height of the column titles displayed in a [Section 6.263](#) object. It is expressed in the units specified by the [Section 6.128](#) property of the [Section 6.263](#). The value 0 selects the default height.

6.619 TitleWidth Property

Applies To: [Section 6.263](#)

Description

This property is a single number that specifies the width of the row titles displayed in a [Section 6.263](#) object. It is expressed in the units specified by the [Section 6.128](#) property of the [Section 6.263](#). The value 0 selects the default width.

6.620 Today Property

Applies To: [Section 6.58](#), [Section 6.143](#)

Description

The Today property is an [Section 8.5](#) that specifies today's date in a [Section 6.58](#) or [Section 6.143](#) object. Its default value is the current date that is set on your computer.

See also [Section 6.95](#) and [Section 6.290](#) properties.

6.621 ToolBar Object

Purpose: To manage a group of controls such as [Section 6.55](#) s.

Parents ActiveXControl CoolBand Form SubForm

Children Bitmap BrowseBox Button Calendar Circle Combo ComboEx Cursor DateTimePicker Edit Ellipse FileBox Font Group Icon Image ImageList Label List ListView Locator Marker Menu Metafile MsgBox OCXClass Poly ProgressBar Rect RichEdit Scroll SM Spinner Static SubForm Text Timer TrackBar TreeView UpDown

Properties Type Posn Size Coord Align Border Active Visible Event VScroll HScroll Sizeable FontObj FCol BCol Picture OnTop IconObj CursorObj AutoConf YRange XRange Data Attach TextSize EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles KeepOnClose Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel Select

Description

The ToolBar object is used to display and manage a set of controls. It is typically used to present a set of [Section 6.55s](#) which the user can press to perform various actions. However, the ToolBar has the ability to manage other controls too.

By default, the ToolBar is a raised bar stretched across the top of its parent form. You can alter its appearance using its [Section 6.184](#) property and you can control its alignment with its [Section 6.13](#) property. [Section 6.13](#) can be set to Top (the default), Bottom, Left or Right and causes the ToolBar to be attached to the corresponding edge of the [Section 6.227](#). A ToolBar aligned Top or Bottom will automatically stretch or shrink horizontally when its parent [Section 6.227](#) is resized, but it will remain fixed vertically. A ToolBar aligned Left or Right will stretch vertically but will remain fixed horizontally. By default a ToolBar occupies the entire width or length of the side of the [Section 6.227](#) to which it is attached and is 30 pixels high or wide. You can change these defaults using the [Section 6.452](#) and [Section 6.556](#) properties.

A ToolBar organises its child controls in the order they are created. The way this is done is governed by the value of the [Section 6.13](#) property. If [Section 6.13](#) is Top or Bottom, the ToolBar arranges its controls in rows across the screen. If [Section 6.13](#) is Left or Right, the ToolBar arranges controls in columns.

The first control added to a ToolBar is automatically positioned 2 pixels down and 2 pixels across from its top left corner. The rule for positioning subsequent controls depends upon the value of the [Section 6.13](#) property.

If [Section 6.13](#) is 'Top' or 'Bottom', controls are positioned so as to be horizontally adjacent to one another. Whenever a control is added it is positioned relative to the one that immediately preceded it so that its top left corner meets the top right corner of the previous one. The [Section 6.303](#) property determines what happens when the end of the ToolBar is reached. If [Section 6.303](#) is 0 (the default) a control that would otherwise extend beyond the width of the ToolBar is instead positioned immediately below the first control in the ToolBar, thereby starting a new row. Note however that the ToolBar is not automatically resized vertically to accommodate a second row. If you want a multi-row ToolBar you have to set its height explicitly. If [Section 6.303](#) is -1 or -2, controls continue to be added along the ToolBar even though they extend beyond its right edge and may be scrolled into view using a mini scrollbar. If [Section 6.303](#) is -1, the scrollbar is shown whether or not any controls extend beyond the width of the ToolBar. If [Section 6.303](#) is -2, the scrollbar appears only if required and may appear or disappear when the user resizes the parent [Section 6.227](#).

If [Section 6.13](#) is 'Left' or 'Right', controls are positioned so as to be vertically adjacent to one another. Whenever a control is added, its top left corner is positioned against the bottom left corner of the previous control. The [Section 6.646](#) property determines what happens when the bottom of the ToolBar is reached. If [Section 6.646](#) is 0 (the default) a control that would otherwise extend beyond the bottom of the ToolBar is instead positioned immediately to the right of the first one; thereby starting a new column. Note however that the ToolBar is not automatically resized horizontally to accommodate a second column. You must set the width of the ToolBar explicitly.

If [Section 6.646](#) is -1 or -2, controls continue to be added down the ToolBar even though they extend beyond its bottom edge and may be scrolled into view using a mini scrollbar. If [Section 6.646](#) is -1, the scrollbar is shown whether or not any controls extend beyond the bottom of the ToolBar. If [Section 6.646](#) is -2, the scrollbar appears only if required and may appear or disappear when the user resizes the parent [Section 6.227](#).

[Section 6.646](#) and [Section 6.303](#) may only be set when the object is created and may not subsequently be changed.

If you specify a value for its [Section 6.452](#) property, a control will be placed at the requested position regardless of the value of [Section 6.577](#), [Section 6.646](#) or [Section 6.303](#). However, the next control added will take its default position from the previous one according to the value of these properties. Thus if you wish to group your controls

together with spaces between the groups, you need only specify the position of the first one in each group.

The `ToolBar` object was introduced in Dyalog APL before an appropriate standard Windows control existed. The `ToolBar` object should be considered as a legacy object and used only in old GUI applications. The [Section 6.624](#) object should be used instead.

6.622 ToolboxBitmap Property

Applies To: [Section 6.7](#), [Section 6.412](#)

Description

For an [Section 6.7](#), the `ToolboxBitmap` property is a character vector or ref that specifies the name of, or ref to, a [Section 6.47](#) object that may be used by a host application to represent the [Section 6.7](#) when its complete visual appearance is not required. For example, if you add an ActiveX control to the Microsoft Visual Basic development environment, its bitmap is added to the toolbox. The Bitmap should therefore be of an appropriate size, usually 24 x 24 pixels.

For an [Section 6.412](#), The `ToolboxBitmap` is a read-only property that reports a bitmap image associated with an OLE Control. This is intended for use by a GUI design tool. Its value is a 2-element vector. The first element is an integer matrix of pixel colours corresponding to the `Bits` property of a `Bitmap` object. The second element is a 3-column integer matrix specifying the colour map and corresponds to the `CMap` property of a `Bitmap` object.

Thus you can construct a `Bitmap` object directly from this property with an expression such as:

```
'BM'[WC'Bitmap' ],'GAUGE' [WG'ToolboxBitmap'
```

where `GAUGE` is the name of an [Section 6.412](#).

6.623 ToolButton Object

Purpose: The `ToolButton` object represents a button in a `ToolControl`.

Parents `ToolControl`

Children `Bitmap` `Timer`

Properties Type Caption Posn Size Style State Active Visible Event ImageIndex Data Hint HintObj Tip Accelerator Popup KeepOnClose MethodList ChildList EventList PropList

Methods Detach

Events Close Create Help MouseDown MouseUp MouseMove MouseDbClick Select

Description

The ToolButton object represents a selectable button in a [Section 6.624](#) object.

A ToolButton displays a text string, defined by its [Section 6.66](#) property, and an image defined by its [Section 6.315](#) property. Apart from these characteristics, the appearance of a ToolButton is controlled by its parent [Section 6.624](#) object.

[Section 6.315](#) is an index into an [Section 6.316](#) which contains a set of icons or bitmaps. The [Section 6.316](#) itself is named by the [Section 6.317](#) property of the parent [Section 6.624](#).

Typically, you will create up to three [Section 6.316](#) as children of the [Section 6.624](#). These will be used to specify the pictures of the ToolButton objects in their normal, highlighted (sometimes termed *hot*) and inactive states respectively. The set of images in each [Section 6.316](#) is then defined by creating unnamed [Section 6.47](#) or [Section 6.309](#) objects as children. Finally, when you create each ToolButton you specify [Section 6.315](#), selecting the three pictures which represent the three possible states of the button.

If you specify only a single [Section 6.316](#), the *picture* on the ToolButton will be the same in all three states.

The behaviour and appearance of a ToolButton is further defined by its [Section 6.577](#) property, which may be 'Push', 'Check', 'Radio', 'Separator' or 'DropDown'.

Push buttons are used to generate actions and pop in and out when clicked. Radio and Check buttons are used to select options and have two states, normal (out) and selected (in). Their [Section 6.571](#) property is 0 when the button is in its normal (unselected state) or 1 when it is selected.

A group of adjacent ToolButtons with [Section 6.577](#) 'Radio' defines a set in which only one of the ToolButtons may be selected at any one time. The act of selecting one will automatically deselect any other. Note that a group of Radio buttons must be separated from Check buttons or other groups of Radio buttons by ToolButtons of another [Section 6.577](#).

A ToolButton with [Section 6.577](#) 'Separator' has no [Section 6.66](#) or picture, but appears as a vertical line and is used to separate groups of buttons.

A ToolButton with [Section 6.577](#) 'DropDown' has an associated popup [Section 6.382](#) object which is named by its [Section 6.451](#) property. There are two cases to consider.

If the [Section 6.548](#) property of the parent [Section 6.624](#) is 0, clicking the ToolButton causes the popup menu to appear. In this case, the ToolButton itself does not itself generate a [Section 6.513](#) event; you must rely on the user selecting a [Section 6.384](#) to specify a particular action.

If the [Section 6.548](#) property of the parent [Section 6.624](#) is 1, clicking the dropdown button causes the popup menu to appear; clicking the ToolButton itself generates a Select event, but does not display the popup menu.

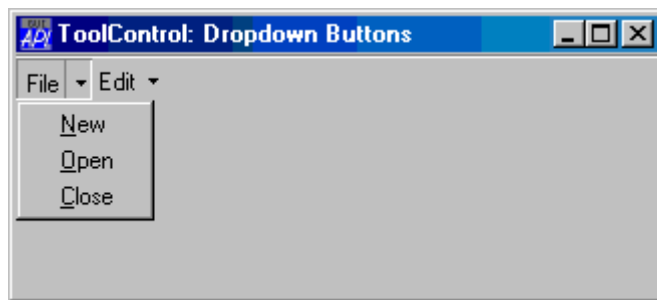
The following example illustrates the use of DropDown buttons:

```
'F'□WC'Form' 'ToolControl: Dropdown Buttons'('Size' 20 40)
'F.TB'□WC'ToolControl'('ShowDropDown' 1)

:With 'F.FMENU'□WC'Menu' A Popup File menu
  'NEW'□WC'MenuItem' '&New'
  'OPEN'□WC'MenuItem' '&Open'
  'CLOSE'□WC'MenuItem' '&Close'
:EndWith

:With 'F.EMENU'□WC'Menu' A Popup File menu
  'CUT'□WC'MenuItem' 'Cu&t'
  'COPY'□WC'MenuItem' '&Copy'
  'PASTE'□WC'MenuItem' '&Paste'
:EndWith

'F.TB.B1'□WC'ToolButton' 'File'('Style' 'DropDown')('Popup' 'F.FMENU')
'F.TB.B2'□WC'ToolButton' 'Edit'('Style' 'DropDown')('Popup' 'F.EMENU')
```



6.624 ToolControl Object

Purpose: The ToolControl object provides a standard Windows ToolBar.

Parents ActiveXControl CoolBand Form SubForm

Children Bitmap BrowseBox Button Combo ComboEx Cursor Edit FileBox Font Group Icon Image ImageList Label List ListView Locator Menu MenuBar Metafile MsgBox OCXClass ProgressBar RichEdit Scroll SM Spinner Static SubForm Timer ToolButton TrackBar TreeView UpDown

Properties Type Posn Size Style Align Visible Event ImageListObj FontObj Data Attach Handle KeepOnClose MultiLine Transparent Divider ShowCaptions ShowDropDown Dockable UndocksToRoot Redraw ButtonsAcceptFocus MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help MouseDown MouseUp MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel DockStart DockMove DockRequest DockAccept DockEnd DockCancel

Description

The ToolControl object provides an interface to the native Windows ToolBar control and supersedes the Dyalog APL [Section 6.621](#) object.

The tools on a ToolControl are normally represented by [Section 6.623](#) objects, but the ToolControl may also act as a parent for other objects, including a [Section 6.383](#) (see below).

Unlike the [Section 6.621](#), the ToolControl fully determines the positioning of its children automatically and this is governed by their order of creation. The Posn property of any child of a ToolControl is therefore read-only. Furthermore, the height of objects in a ToolControl may be no greater than that of a [Section 6.623](#) in the same ToolControl. This in turn is governed by the sizes of the [Section 6.225](#) and [Section 6.316](#) in use in that ToolControl. ToolControl objects should be used in preference to [Section 6.621](#) objects.

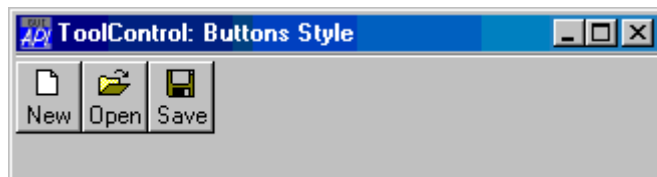
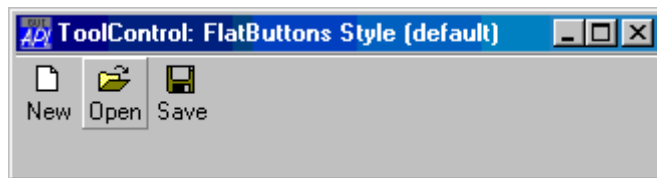
If a ToolControl is the child of a [Section 6.227](#), its position and orientation is defined by its [Section 6.13](#) property. This property is ignored if the ToolControl is the child of a [Section 6.126](#).

The overall appearance of the [Section 6.623](#) objects displayed by the ToolControl is defined by the [Section 6.577](#) property of the ToolControl itself, rather than by individual [Section 6.623](#). This may be 'Buttons', 'FlatButtons', 'List' or 'FlatList'.

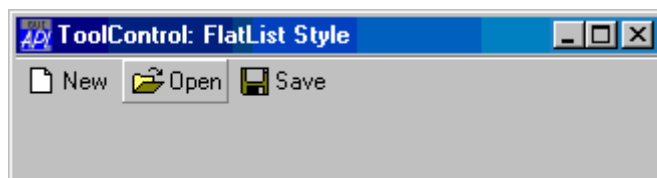
```
'F'WC'Form' 'ToolControl: FlatButtons Style (default)'('Size' 10 40)
'F.TB'WC'ToolControl'

'F.TB.IL'WC'ImageList'('Masked' 0)
'F.TB.IL.'WC'Bitmap'('Comctl32' 120)
'F.TB'WS'ImageListObj' 'F.TB.IL'

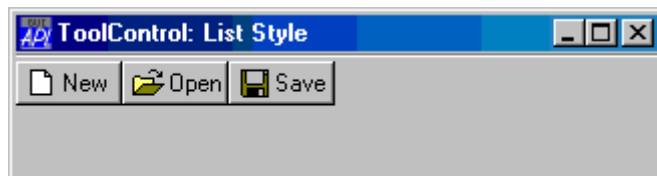
'F.TB.B1'WC'ToolButton' 'New'('ImageIndex' 7)
'F.TB.B2'WC'ToolButton' 'Open'('ImageIndex' 8)
'F.TB.B3'WC'ToolButton' 'Save'('ImageIndex' 9)
```



```
'F.TB'WC'ToolControl'('Style' 'Buttons')
```



```
'F.TB'WC'ToolControl'('Style' 'FlatList')
```



```
'F.TB'WC'ToolControl'('Style' 'List')
```

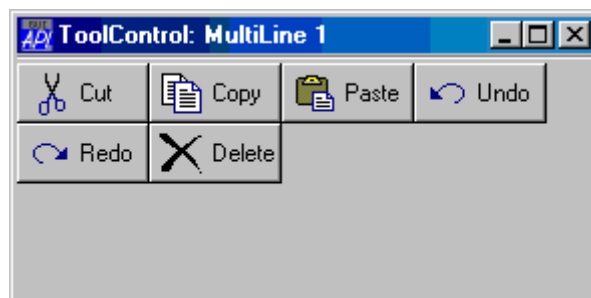
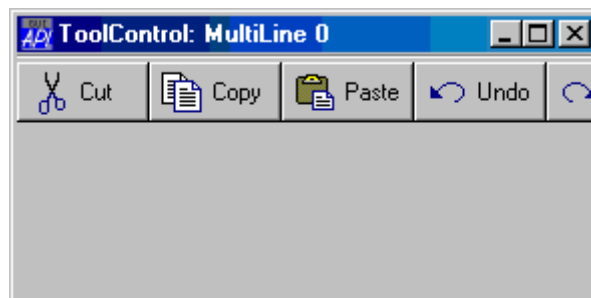
The presence or absence of a recessed line drawn above, below, to the left of, or to the right of the ToolControl is controlled by the [Section 6.161](#) property whose default is 1 (show divider).

The [Section 6.404](#) property specifies whether or not [Section 6.623](#) (and other controls) are arranged in several rows (or columns) when there are more than will otherwise fit. If [Section 6.404](#) is 0 (the default), the ToolControl object clips its children and the user must resize it to bring more objects into view.

```
'F'WC'Form' 'ToolControl: MultiLine 0'('Size' 20 36)
'F.TB'WC'ToolControl'('Style' 'List')

'F.TB.IL'WC'ImageList'('Masked' 0)('Size' 24 24)
'F.TB.IL.'WC'Bitmap'('ComCtl32' 121)A STD_LARGE
'F.TB'WS'ImageListObj' 'F.TB.IL'

'F.TB.B1'WC'ToolButton' 'Cut'('ImageIndex' 1)
'F.TB.B2'WC'ToolButton' 'Copy'('ImageIndex' 2)
'F.TB.B3'WC'ToolButton' 'Paste'('ImageIndex' 3)
'F.TB.B4'WC'ToolButton' 'Undo'('ImageIndex' 4)
'F.TB.B5'WC'ToolButton' 'Redo'('ImageIndex' 5)
'F.TB.B6'WC'ToolButton' 'Delete'('ImageIndex' 6)
```



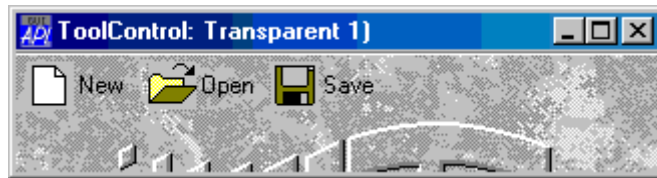
```
'F.TB'WC'ToolControl'('MultiLine' 1)('Style' 'List')
```

The [Section 6.628](#) property specifies whether or not the ToolControl is transparent. If so, the visual effect is as if the [Section 6.623](#) (and other controls) were drawn directly on the parent Form as illustrated below:

```
'F'□WC'Form' 'ToolControl: Transparent 1)'('Size' 10 40)
'F.BM'□WC'Bitmap' 'C:\WINDOWS\WINLOGO'
'F'□WS'Picture' 'F.BM' 1

'F.TB'□WC'ToolControl'('Transparent' 1)('Style' 'FlatList')
'F.TB.IL'□WC'ImageList'('Masked' 0)('Size' 24 24)
'F.TB.IL.'□WC'Bitmap'('ComCtl32' 121)A STD_LARGE
'F.TB'□WS'ImageListObj' 'F.TB.IL'

'F.TB.B1'□WC'ToolButton' 'New'('ImageIndex' 7)
'F.TB.B2'□WC'ToolButton' 'Open'('ImageIndex' 8)
'F.TB.B3'□WC'ToolButton' 'Save'('ImageIndex' 9)
```



The [Section 6.544](#) property specifies whether or not the captions of [Section 6.623](#) objects are drawn. Its default value is 1 (draw captions). [Section 6.623](#) drawn without captions occupy much less space and [Section 6.544](#) provides a quick way to turn captions on/off for user customisation.

The [Section 6.548](#) property specifies whether or not a drop-down menu symbol is drawn alongside [Section 6.623](#) which have [Section 6.577](#) 'DropDown'. [Section 6.548](#) also affects the behaviour of such [Section 6.623](#) objects when clicked.

The [Section 6.57](#) property determines how the ToolControl responds to the Tab and cursor movement keys.

As a special case, the ToolControl may contain a [Section 6.383](#) as its **only** child. In this case, Dyalog APL causes the menu items to be drawn as buttons as shown below.

Although nothing is done to prevent it, the use of other objects in a ToolControl containing a [Section 6.383](#), is not supported.

```

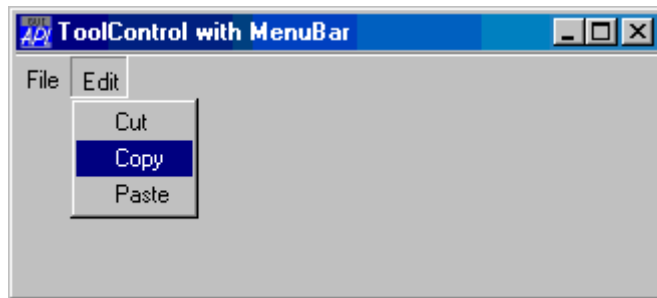
'F'WC'Form' 'ToolControl with MenuBar'('Size' 20 40)
'F.TB'WC'ToolControl'

:With 'F.TB.MB'WC'MenuBar'
  :With 'File'WC'Menu' 'File'
    'New'WC'MenuItem' 'New'
    'Open'WC'MenuItem' 'Open'
    'Close'WC'MenuItem' 'Close'
  :EndWith

  :With 'Edit'WC'Menu' 'Edit'
    'Cut'WC'MenuItem' 'Cut'
    'Copy'WC'MenuItem' 'Copy'
    'Paste'WC'MenuItem' 'Paste'
  :EndWith

:EndWith

```



6.625 TrackBar Object

Purpose: The TrackBar object is a slider control that allows the user to enter a value by positioning a pointer (thumb) on a scale.

Parents ActiveXControl CoolBand Form Grid Group PropertyPage SubForm ToolBar
ToolControl

Children Bitmap Circle Cursor Ellipse Font Icon Marker Poly Rect Text Timer

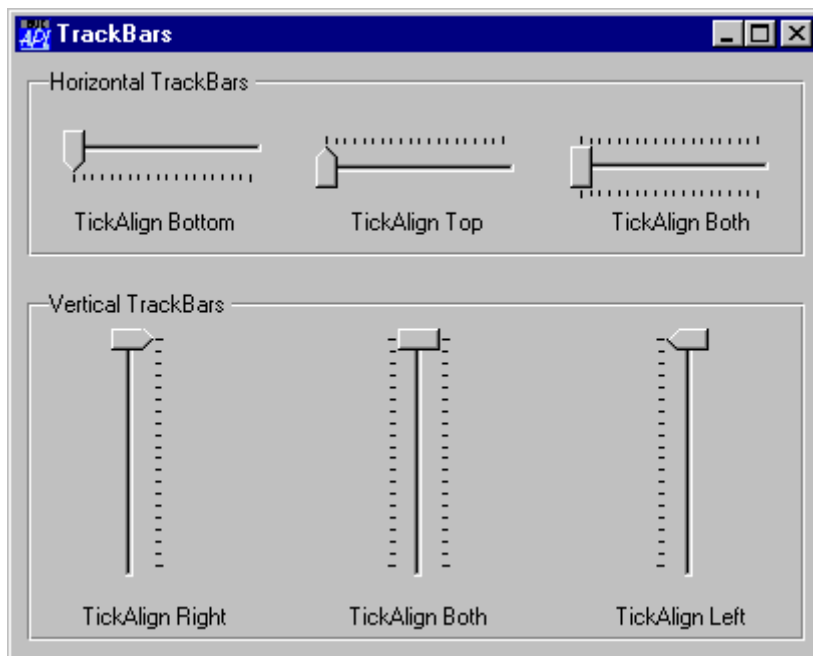
Properties Type Posn Size Style Coord Border Active Visible Event Thumb Step VScroll
HScroll Limits SelRange Sizeable Dragable FontObj BCol CursorObj AutoConf Data
Attach EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator TickAlign
TickSpacing HasTicks ShowThumb TrackRect ThumbRect AcceptFiles KeepOnClose
Redraw TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp MouseMove Select ThumbDrag Scroll

Description

The `TrackBar` object consists of a window which contains a slider bar, a thumb, and a set of tick marks. 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 of five, the slider can only occupy six positions: a position at the left side of the `TrackBar` and one position for each increment in the range. Typically, each of these positions is identified by a tick mark. `TrackBars` can have either a vertical or horizontal orientation. They can have tick marks on either side, both sides, or neither. A selection of different `TrackBars` is illustrated below.



The position and size of the container window are defined by the [Section 6.452](#) and [Section 6.556](#) properties. Its appearance is defined by the [Section 6.184](#), [Section 6.49](#) and [Section 6.45](#) properties. The defaults are ('EdgeStyle' 'None'), ('Border' 0) and ('BCol' 0). The default background colour ('BCol' 0) obtains either the standard Window Background colour, or grey to match the colour of the parent object if it has a 3-dimensional appearance.

The orientation of a `TrackBar` is determined by the [Section 6.303](#) and [Section 6.646](#) properties. A horizontal `TrackBar` is obtained by setting [Section 6.303](#) to `-1` and [Section 6.646](#) to `0`. This is the default. A vertical `TrackBar` is obtained by setting [Section 6.646](#) to `-1` and [Section 6.303](#) to `0`.

[Section 6.646](#) and [Section 6.303](#) may only be set when the object is created and may not subsequently be changed.

The [Section 6.554](#) property determines whether or not the thumb is visible. Its default value is `1`. You may toggle this property dynamically using `WS`.

The `TrackBar` optionally displays tick marks at the two ends of the slider bar and spaced out along it. This behaviour is determined by the [Section 6.289](#) property which may be `1` (the default) or `0` and may be set *only* when the object is created by `WC`.

If [Section 6.289](#) is `1`, the position and frequency of the tick marks is determined by the [Section 6.611](#) and [Section 6.612](#) properties. Note that [Section 6.611](#) may only be set when the `TrackBar` is created with `WC` and may not be altered using `WS`.

The slider and tick marks in a horizontal `TrackBar` are drawn along the top of the enclosing window. The slider and tick marks in a vertical `TrackBar` are drawn along the left edge of the window. The position and size of the slider and the thumb may be obtained from the [Section 6.626](#) and [Section 6.610](#) properties which report these values in pixels. These are *read-only* properties and cannot not be set with `WC` or `WS`.

The value of the `TrackBar` is determined by its [Section 6.608](#) property which is an *integer* that may be set with `WS` or retrieved with `WC`. The [Section 6.345](#) property specifies the minimum and maximum values of Thumb corresponding to its position at the two ends of the slider bar. The [Section 6.576](#) property is a 2-element integer vector defining the small and large increments by which the Thumb moves. A small step is obtained by pressing a cursor movement key; a large step is achieved by clicking the left mouse button either side of the thumb or by pressing Page Up and Page Down. The user may also drag the thumb to a new position or move it directly to either end of the slider by pressing Home or End.

An alternative form of the `TrackBar` is obtained by setting the [Section 6.577](#) property to `'Selection'`. This may only be done when the object is created using `WC`. This style of `TrackBar` has a slider that is represented by a recessed thick white rectangle instead of a solid black line. Furthermore, you can select a range of values within the `TrackBar` by setting the [Section 6.520](#) property. This causes the `TrackBar` to display a solid blue bar within the white slider and to show the corresponding tick marks as small triangles. Note that there is no way for the user to change [Section 6.520](#) directly; you can only do this using `WS`.

In addition to the normal mouse events, the `TrackBar` generates a [Section 6.509](#) and [Section 6.609](#) event. The [Section 6.509](#) event is the same event that is generated by a `Scroll` object and is reported when the user repositions the thumb. If enabled, the [Section 6.609](#) event is reported continuously as the user drags the thumb with the mouse and may be used to synchronise the display of a corresponding value in another object.

6.626 TrackRect Property

Applies To: [Section 6.625](#)

Description

`TrackRect` is a *read-only* property that reports the position and size of the bounding rectangle of the slider in a [Section 6.625](#) object. It is a 4-element integer vector containing:

[1]	Vertical position of the top-left corner of the bounding rectangle
[2]	Horizontal position of the top-left corner of the bounding rectangle
[3]	Height of the bounding rectangle
[4]	Width of the bounding rectangle

6.627 Translate Property

Applies To

Description

This property specifies whether or not character data is to be translated.

`Translate` applies to the Classic Edition only. In the Unicode Edition, its value is ignored.

`Translate` is a character vector whose values may be `'Inherit'`, `'Translate'`, `'ANSI'` or `'None'`.

A value of `'Translate'` means that all character property values and event parameters are translated to and from `␣V` using the current output translation table (`WIN.DOT`).

`'None'` means that character data is passed between APL and the object with no translation.

If you set the value of the Translate property to 'ANSI', APL does not attempt to resolve characters as they are typed by the user via the Input Translate Table. Using Translate 'ANSI' in combination with the appropriate value of [Section 6.88](#) and the corresponding National Language keyboard will permit users to enter strings in non-western languages.

'Inherit' means that the object inherits its translation from its parent.

The default value for the Root and Printer objects is 'Translate', and for most other objects it is 'Inherit'.

6.628 Transparent Property

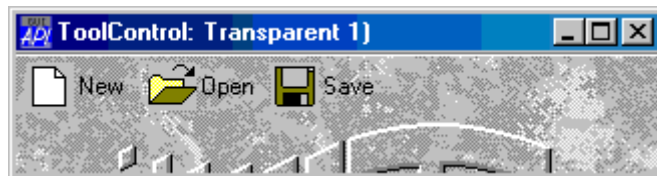
Applies To: [Section 6.21](#), [Section 6.56](#), [Section 6.624](#)

Description

The Transparent property specifies whether or not a [Section 6.624](#) is transparent.

Transparent is a single number with the value 0 (the default) or 1.

If Transparent is 1, the visual effect is as if the [Section 6.623](#) (and other controls owned by the [Section 6.624](#)) were drawn directly on the parent Form as illustrated below.



6.629 TreeView Object

Purpose: The TreeView object displays a hierarchical list of items.

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm ToolBar ToolControl

Children Bitmap Circle Cursor Ellipse Font Icon ImageList Marker Poly Rect Text Timer

Properties Type Items Posn Size Coord Border Active Visible Event Depth HasLines HasButtons EditLabels ImageListObj ImageIndex SellImageIndex SellItems Sizeable Draggable FontObj FCol BCol CursorObj AutoConf Index Data Attach EdgeStyle Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles KeepOnClose CheckBoxes

FullRowSelect SingleClickExpand Redraw TabIndex AlwaysShowSelection MethodList
ChildList EventList PropList

Methods Detach ChooseFont GetItemState SetItemState AddItems DeleteItems
AddChildren DeleteChildren GetParentItem GetItemHandle SetItemImage ShowItem
GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create FontOK FontCancel DragDrop Configure ContextMenu DropFiles
DropObjects Expose Help KeyPress GotFocus LostFocus MouseDown MouseUp
MouseMove MouseDbClick MouseEnter MouseLeave MouseWheel GestureZoom
GesturePan GestureRotate GestureTwoFingerTap GesturePressAndTap BeginEditLabel
EndEditLabel Expanding Retracting ItemDown ItemUp ItemDbClick GetTipText Select

Description

A TreeView object 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 bitmapped image, and each item can have a list of sub-items associated with it. By clicking an item, the user can expand and collapse the associated list of sub-items.

The contents of a TreeView object are defined by the [Section 6.334](#) property; a vector of character vectors that specifies the item labels.

The [Section 6.317](#), [Section 6.315](#) and [Section 6.518](#) properties define bitmapped images corresponding to each item. The bitmapped images are drawn to the left of the item labels.

[Section 6.317](#) specifies the name of a single [Section 6.316](#) object that contains one or more bitmaps.

[Section 6.315](#) and [Section 6.518](#) are $\square$ IO sensitive scalars, or vectors with the same length as the number of items in the object. The value in the *i*th element specifies the image for the *i*th item and is an index into the corresponding [Section 6.316](#) object. [Section 6.315](#) specifies the image displayed when an item is *not selected*, [Section 6.518](#) specifies the image displayed when an item is *selected*.

If [Section 6.317](#) is specified, but ImageIndex is empty or not specified, the first bitmap in the [Section 6.316](#) is drawn alongside every item. If an element of [Section 6.315](#) or [Section 6.518](#) specifies a value that does not correspond to a bitmap in the [Section 6.316](#), no picture is drawn.

The structure of the items (that is, the parent/child relationships of the items) is defined by the [Section 6.156](#) property. This is either a scalar 0 (the default) which

means that all items are *root* items, or it is a numeric vector of the same length as Items. Non-zero values in [Section 6.156](#) indicate child items.

The [Section 6.288](#) property is 0, 1 or 2 and determines whether or not lines are drawn that link child items to their corresponding parent item. If [Section 6.288](#) is 0, no lines are drawn. If [Section 6.288](#) is 1, lines are drawn at all except the top level, that is, the object does not link items at the root of the hierarchy. The default value for [Section 6.288](#) is 2 which provides lines at all levels including the root.

The [Section 6.283](#) property determines whether or not the TreeView object has a button to the left side of each parent item. It is Boolean with a default value of 1. The user can click the button to expand or collapse the child items as an alternative to double-clicking the parent item. Note that by itself, setting [Section 6.283](#) to 1 does not add buttons to items at the root of the hierarchy. To achieve this you must also set [Section 6.288](#) to 2.

The [Section 6.89](#) property specifies whether or not check boxes are displayed alongside items in a TreeView.

The [Section 6.232](#) property specifies whether just the item itself, or the entire row of the TreeView, is highlighted when an item is selected. [Section 6.232](#) should not be used if [Section 6.288](#) is 1 or 2

When the user presses the left mouse button over an item, the object generates an [Section 6.331](#) event. This is followed by an [Section 6.335](#) event when the mouse button is released. The object also generates an [Section 6.330](#) event when the left mouse is double-clicked over an item. If all three events are enabled, they are reported in the order [Section 6.331](#), [Section 6.330](#), [Section 6.335](#).

When a parent item is in its retracted state (its children are not visible) it can be expanded to show its children by the user double-clicking its label or by clicking over its button or tree lines. An [Section 6.201](#) event is reported immediately before the children are shown. Similarly, when a parent item is in its expanded state, it can be retracted to hide its children when a [Section 6.487](#) event is reported. You can use the [Section 6.201](#) event to define new children for the object just before they are shown. You can also control the actions of these events using callback functions.

The [Section 6.188](#) is a Boolean property (default 0) that determines whether or not the user may edit the labels which are specified by the Items property.

The [Section 6.519](#) property is a Boolean vector that indicates which of the items is currently selected and has the focus. If more items are visible than can fit within the object, a scrollbar is automatically provided. The [Section 6.319](#) property is a ☐IO

sensitive integer that reports the index number of the first item displayed in the object and changes as the items are scrolled.

Warning

Due to the limitations of the Win32 TreeView object, it is necessary to query the state of each item in a TreeView in order to obtain the value of the [Section 6.519](#) property, making it a comparatively slow operation if there are a lot of [Section 6.334](#).

6.630 Type Property

Applies To

Description

This property determines the type of an object. Its value is a character vector containing a valid object type. The Type property is set by *Dyalog APL Language: Wc* and reported by *Dyalog APL Language: Wg*, but may not be altered using *Dyalog APL Language: Ws*.

6.631 TypeLibFile Property

Applies To: [Section 6.7](#), [Section 6.420](#)

Description

The TypeLibFile property is a read-only property that reports the name of the file in which the Type Library associated with a COM object is stored.

6.632 TypeLibID Property

Applies To: [Section 6.7](#), [Section 6.420](#)

Description

The TypeLibID property is a read-only property that reports the value of the globally unique identifier ([Section 8.4](#)) of the Type Library associated with a COM object.

6.633 TypeList Property

Applies To: [Section 6.412](#), [Section 6.414](#)

Description

This property reports the names of all the special data types defined for a particular COM object. It is a vector of character vectors returned by `TYPE`. It may not be set using `TYPE` or `WS`. Further information about each data type may be obtained using [Section 6.258](#).

Note that TypeList reports **all** of the data type names recorded in the .OCX file associated with the COM object. If several COM objects are provided within a single .OCX file, the entire set of data types reported may not necessarily be applicable to the Control in question.

6.634 Underline Property

Applies To: [Section 6.222](#)

Description

This property specifies whether or not the characters in the font associated with a [Section 6.222](#) object are underlined or not. It is either 0 (normal) or 1 (underlined). There is no default; the value of this property reflects the underline characteristic of the font allocated by Windows.

6.635 Undo Method 170

Applies To: [Section 6.263](#)

Description

This method is used to undo the previous change in a [Section 6.263](#) object.

The [Section 6.263](#) object maintains a buffer of the most recent 8 changes made by the user since the [Section 6.642](#) property was last set by *Dyalog APL Language: Wc* or *Dyalog APL Language: Ws*.

Your application can restore these changes one by one by calling the Undo method on the [Section 6.263](#). The Undo method restores the most recent change made by the user and removes that change from the undo stack.

It is therefore not possible to "undo an undo".

The argument to Undo is $\emptyset$, or a single item as follows :

[1]	Number of changes	integer

If called with an argument of $\emptyset$, the default value for the *Number of changes* is 1. This restores the most recent change.

6.636 UndocksToRoot Property

Applies To: [Section 6.126](#), [Section 6.227](#), [Section 6.578](#), [Section 6.624](#)

Description

Specifies the parent adopted by an object when its Type changes to a Form as a result of an undocking operation.

UndocksToRoot is a single number with the value 0 or 1.

If UndocksToRoot is 1, the object becomes a Form that is a child of Root and therefore becomes completely independent of the Form in which it was previously docked.

If UndocksToRoot is 0, the object becomes a Form that is a child of the Form in which it was previously docked and is therefore always displayed on top of it. This setting is appropriate for a dockable toolbar.

The default value of UndocksToRoot is 1 if the object was originally created as a child of Root; otherwise it is 0.

6.637 UpDown Object

Purpose: The UpDown object is a pair of arrow buttons used to increment or decrement a value.

Parents ActiveXControl CoolBand Form Group PropertyPage SubForm ToolBar ToolControl

Children Bitmap Circle Cursor Ellipse Font Icon Marker Poly Rect Text Timer

Properties Type Posn Size Coord Border Active Visible Event Thumb Step VScroll HScroll Wrap Limits Sizeable Draggable FCol BCol CursorObj AutoConf Data Attach EdgeStyle

Handle Hint HintObj Tip TipObj Translate Accelerator AcceptFiles KeepOnClose Redraw
TabIndex MethodList ChildList EventList PropList

Methods Detach GetTextSize Animate GetFocus ShowSIP GetFocusObj

Events Close Create DragDrop Configure ContextMenu DropFiles DropObjects Expose
Help MouseDown MouseUp MouseMove MouseEnter MouseLeave Select Spin

Description

An UpDown object is a pair of arrow buttons that the user can click to increment or decrement a value, such as a scroll position or a number displayed in a companion control. The [Section 6.563](#) object is actually a composite object consisting of an UpDown and a companion Edit.

6.638 URL Property

Applies To: [Section 6.307](#)

Description

The URL property is a character vector that specifies the url of a resource to be requested by the [Section 6.307](#). Its default value is an empty character vector.

Specifying URL is an alternative way to setting the [Section 6.306](#) property in order to display content in the [Section 6.307](#).

When you set the URL property, the [Section 6.307](#) will request the corresponding resource (from either the internet or the workspace via an [Section 6.308](#) event) and the display will change according to the response. The [Section 6.306](#) property is ignored and remains unchanged.

When you set the [Section 6.306](#) property, the content of the [Section 6.307](#) will change accordingly. The current value of the URL property is ignored and remains unchanged.

If you set BOTH URL and [Section 6.306](#) in the same statement, the value of URL takes precedence and the assignment to [Section 6.306](#) is ignored (it remains unchanged).

6.639 ValidIfEmpty Property

Applies To: [Section 6.56](#), [Section 6.185](#), [Section 6.563](#)

Description

This property applies to an [Section 6.185](#) object with [Section 6.577](#) Single and specifies whether or not an empty field is considered to be valid. It also applies to a [Section 6.563](#). Its value is either 0 (an empty field is not valid) or 1 (an empty field is valid). If the [Section 6.206](#) is Numeric, LongNumeric, Currency, Date or Time, the default value for ValidIfEmpty is 0. Otherwise, its default value is 1.

If ValidIfEmpty is 0 and the user attempts to *leave* the [Section 6.185](#) object by shifting the input focus to another control, or by selecting a [Section 6.55](#) or [Section 6.384](#), the [Section 6.185](#) object will generate a [Section 6.38](#) event. The [Section 6.606](#) property will reflect the appearance of the field and be empty, but the [Section 6.641](#) property will not be changed.

If ValidIfEmpty is 1 and the [Section 6.206](#) is Numeric, LongNumeric, Currency, date or Time, the [Section 6.641](#) property will be set to 0 when the user clears the field and leaves it.

6.640 VAlign Property

Applies To: [Section 6.605](#)

Description

This property determines the vertical alignment of text in the [Section 6.605](#) object. It is either a single integer value, or, if the [Section 6.605](#) object has several components, a corresponding vector of such values.

These may be :

0	base aligned	the base line of the character is aligned on the y-coordinate specified by the Points property.
1	half aligned	the centre of the character is aligned on the y-coordinate specified by the Points property.
2	cap aligned	the top of the character is aligned on the y-coordinate specified by the Points property.
3	bottom aligned	the bottom of the character cell is aligned on the y-coordinate specified by the Points property.
4	top aligned	the top of the character cell is aligned on the y-coordinate specified by the Points property. This is the default.

6.641 Value Property

Applies To: [Section 6.56](#), [Section 6.185](#), [Section 6.341](#), [Section 6.563](#)

Description

This property specifies or reports the numeric value associated with an [Section 6.185](#) or [Section 6.341](#) object whose [Section 6.206](#) property is set to Numeric, LongNumeric, Date, LongDate or Time.

If the [Section 6.206](#) is Numeric or LongNumeric, the Value property contains a scalar number. If the [Section 6.206](#) is Date or LongDate, the Value property is an integer representing the date as the number of days since 1st January 1900. If the [Section 6.206](#) is Time, the Value property is an integer that contains the number of seconds since midnight.

6.642 Values Property

Applies To: [Section 6.263](#)

Description

This property specifies the data values for the cells in a [Section 6.263](#) object. Values must be a matrix whose elements are either single numbers, character scalars, character vectors or character matrices. This property is updated as the user moves around the [Section 6.263](#) changing data.

6.643 VariableHeight Property

Applies To: [Section 6.127](#)

Description

The VariableHeight property specifies whether or not a [Section 6.127](#) displays bands at the minimum required height, or all the same height (that of the largest).

VariableHeight is a single number with the value 0 (same height) or 1 (variable height). The default is 1.

6.644 View Property

Applies To: [Section 6.348](#)

Description

The View property specifies how the items in a [Section 6.348](#) object are displayed. It is a character vector which may have one of the following values; 'Icon' (the default), 'SmallIcon', 'List' or 'Report'.

When View is 'Icon' or 'SmallIcon', the items are arranged row-wise with large or small icons as appropriate. When View is set to 'List', the items are arranged column-wise using small icons. When View is set to 'Report', the items are displayed in a single column using small icons but with the matrix specified by [Section 6.480](#) displayed alongside. In this format, the ListView also provides column headings which are specified by the ColTitles property. The alignment of these titles (and of the data in the columns beneath them) is defined by the [Section 6.114](#) property. Examples of different views are illustrated below.





A screenshot of a software window titled 'ListView: View=Report'. It displays a detailed report table for the same 12 countries. The table has five columns: Country, Capital, Population, Area (sq km), and Pop per sq km. Each row includes a small flag icon next to the country name.

Country	Capital	Population	Area (sq km)	Pop per sq km
Australia	Canberra	20155130	7741220	3
Brazil	Brasilia	186404900	8514877	22
Canada	Ottawa	32268240	9970610	3
Denmark	Copenhagen	5430590	43094	126
Finland	Helsinki	5249060	338145	16
France	Paris	60495540	551500	110
Germany	Berlin	82689210	357022	232
Ireland	Dublin	4147901	70273	59
Italy	Rome	58092740	301318	193
Japan	Tokyo	128084700	377873	339
United Kingdom	London	59667840	242900	246
United States	Washington	298212900	9629091	31

6.645 Visible Property

Applies To

Description

This property specifies whether or not an object is currently visible. It is a single number with the value 0 (object is invisible) or 1 (object is visible). The default is 1. Setting Visible on and off is a way to pop a dialog box up and down as required.

Note that an invisible object is not necessarily inactive, and is capable of generating events. For example, a [Section 6.55](#) with a [Section 6.64](#) property of 1 will generate a [Section 6.513](#) (30) event (if enabled) whether or not it is visible. An invisible object will also respond to methods and events sent to it by *Dyalog APL Language: Nq*.

6.646 VScroll Property

Applies To

Description

This property determines whether or not an object has a vertical scrollbar. It is a single integer with the value -2, -1, or 0.

VScroll may only be set when the object is created with `□WC` and may not subsequently be changed with `□WS` or assignment.

For a [Section 6.227](#) object, the value -1 specifies that the [Section 6.227](#) has a vertical scrollbar. A value of 0 (which is the default) means that it does not.

When applied to an [Section 6.185](#) object, the value -2 specifies that the data is scrollable vertically, but only by using the cursor keys; a scrollbar is not provided. A value of -1 causes a scrollbar to be displayed (whether or not one is needed).

When applied to a [Section 6.346](#) object, the value -2 specifies that the data is scrollable vertically, but only by using the cursor keys; a scrollbar is not provided. A value of -1 causes a scrollbar to be displayed if required (when the list of items exceeds the height of the object).

When applied to a [Section 6.121](#) or [Section 6.122](#) object, a value of -1 or -2 causes a scrollbar to be displayed, whether or not one is required.

For all these objects, a value of 0 inhibits scrolling altogether.

For a [Section 6.508](#) object, VScroll may be -1 or 0. If it is -1 the direction of the scrollbar is vertical. If both [Section 6.303](#) and VScroll are set to -1, [Section 6.303](#) takes precedence and forces VScroll back to 0.

For a [Section 6.574](#), [Section 6.582](#) or [Section 6.621](#) with [Section 6.13](#) set to Left or Right, VScroll determines whether or not a vertical scrollbar is provided and how the object

positions its children. If VScroll is 0 (the default) the object organises its children in multiple columns and does not provide a scrollbar. If VScroll is $\bar{1}$ or $\bar{2}$, the object organises its children in a single column and provides a mini scrollbar to allow those positioned beyond the bottom edge of the object to be scrolled into view. If VScroll is $\bar{1}$, the scrollbar is always shown. If VScroll is $\bar{2}$, it is only shown when needed.

For a [Section 6.263](#), VScroll may be 0 (no vertical scrollbar), $\bar{1}$ (scrollbar is displayed when required), $\bar{2}$ (same as $\bar{1}$) or $\bar{3}$ (scrollbar is always displayed).

6.647 VScroll Event 38

Applies To: [Section 6.227](#), [Section 6.578](#)

Description

If enabled, this event is generated when the user attempts to move the thumb in a vertical scrollbar in a [Section 6.227](#) or [Section 6.578](#). This event occurs only in a [Section 6.227](#) whose [Section 6.646](#) property is set to $\bar{1}$ and is distinct from the [Section 6.509](#) event which is generated by a [Section 6.508](#) object. The event may be generated in one of three ways :

1. dragging the thumb
2. clicking in one of the "arrow" buttons situated at the ends of the scrollbar. This is termed a small change, the size of which is defined by [Section 6.576\[1\]](#).
3. clicking in the body of the scrollbar. This is termed a large change, the size of which is defined by [Section 6.576\[2\]](#).

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 4-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'VScroll' or 38
[3]	Scroll Type	numeric
[4]	Position	numeric

The value of Scroll Type is 0 (drag), 1 or $\bar{1}$ (small change) or 2 or $\bar{2}$ (large change). The sign indicates the direction.

The value of Position is the new (requested) position of the thumb. Notice however that the event is generated **before** the thumb is actually moved. If your callback function returns a scalar 0, the position of the thumb will remain unaltered.

6.648 VThumbDrag Event 441

Applies To: [Section 6.227](#), [Section 6.578](#)

Description

If enabled, this event is generated when the user attempts to drag the thumb in a vertical scrollbar in a [Section 6.227](#) or [Section 6.578](#). This event occurs only in a [Section 6.227](#) or [Section 6.578](#) whose [Section 6.303](#) property is set to `1` and is distinct from the Scroll event that is generated by a [Section 6.508](#) object.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 3-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'VThumbDrag' or 441
[3]	Position	numeric

The value of Position is the new (requested) position of the Thumb. Setting the action code of this event to `1`, or returning a 0 from a callback function attached to it, has no effect.

6.649 Wait Method 147

Applies To

Description

The Wait method is the same as executing `⎕DQ` on the object.

The Wait method is niladic.

```
'F'⎕WC'Form'
...
Z←F.Wait
```

6.650 WantsReturn Property

Applies To: [Section 6.185](#), [Section 6.488](#)

Description

This Boolean property specifies the behaviour of the Enter key for a multi-line [Section 6.185](#) (Style 'Multi') and a [Section 6.488](#) object.

A value of 0 means that the Enter key is ignored by the [Section 6.185](#) or [Section 6.488](#). Instead, the Enter key will (if appropriate) cause a Select event on a Button in the same Form. The user must press Ctrl+Enter to input a new line.

A value of 1 means that pressing the Enter key will introduce a new line into the object.

WantsReturn must be established when the object is created by `□wc` and may not subsequently be altered using `□ws`. Its default value is 0 in an Edit and 1 in a RichEdit.

6.651 WebSocketClose Event 843

Applies To: [Section 6.307](#)

Description

This event is triggered when the [Section 6.307](#) client closes the WebSocket. It is for notification only.

The event message reported as the result of `□DQ`, or supplied as the right argument to your callback function, is a 5-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'WebSocketClose' or 843
[3]	ID	Character vector containing the ID of the WebSocket
[4]	Status Code	Integer. 1000 indicates normal closure. See https://www.iana.org/assignments/websocket/websocket.xml#close-code-number .
[5]	Reason	Character vector, maximum length 123 bytes.

When called as a method, the result is 0.

EXAMPLE

```
hr.WebSocketClose '223d0f781e95113'
```

6.652 WebSocketError Event 844

Applies To: [Section 6.307](#)

Description

This event is triggered an error occurs on the WebSocket. It is for notification only.

The event message reported as the result of `onerror`, or supplied as the right argument to your callback function, is a 4-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'WebSocketError' or 844
[3]	ID	Character vector containing the ID of the WebSocket
[4]	Message	The error message (character vector)

6.653 WebSocketReceive Event 842

Applies To: [Section 6.307](#)

Description

This event is triggered when data is received over a WebSocket. This event is reported for information only. The result (if any) of a callback function will be ignored.

Note that the WebSocket protocol provides for the possibility for the data to be sent in chunks, causing a succession of WebSocketReceive events. The FIN bit of the last chunk will be 1. The CEF does not currently implement "chunking", so FIN will always be 1.

The event message reported as the result of `onmessage`, or supplied as the right argument to your callback function, is a 6-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'WebSocketReceive' or 842
[3]	ID	Character vector containing the ID of the WebSocket
[4]	Data	Character or integer vector.
[5]	FIN	Boolean. 1 indicates that this is the last chunk; 0 that there is more to come.
[6]	Datatype	1 = character, 2 - numeric values in the range ~128 to 127.

EXAMPLE



6.654 WebSocketSend Method 847

Applies To: [Section 6.307](#)

Description

This method is used to send data to a WebSocket. The argument to `WebSocketSend` is a 2, 3 or 4-element vector as follows:

[1]	ID	Character vector containing the ID of the WebSocket
[2]	Data	Character or integer vector. Integers must be in the range ~128 to 255.
[3]	FIN	Boolean. 1 indicates that this is the last chunk; 0 that there is more to come. This is not currently supported by the CEF and should be set to 1.
[4]	OpCode	Integer. 1 means UTF-8 data, 2 means binary data. Should be 0 if FIN is 0.

The result is 0.

EXAMPLE

```
hr.WebSocketSend '5d61d8330065608' 'Hello World'
0
```

6.655 WebSocketUpgrade Event 841

Applies To: [Section 6.307](#)

Description

This event is reported when the client component of an [Section 6.307](#) object opens a WebSocket and the requested URL matches a pattern specified by the [Section 6.326](#) property. If there is no match, the connection request is processed as an external request by the [Chromium Embedded Framework \(CEF\)](#).

The event message reported as the result of `□□□`, or supplied as the right argument to your callback function, is a 6-element vector as follows:

[1]	Object	ref or character vector
[2]	Event	'WebSocketUpgrade' or 841
[3]	ID	Character vector containing the ID of the WebSocket
[4]	URL	The requested URL of the WebSocket
[5]	Headers	ASCII including CRLF
[6]	Type	Character vector 'auto' or 'manual '

The protocol for establishing the connection is defined by [Section 6.326](#) and is reported by the 6th element (Type) of the event message.

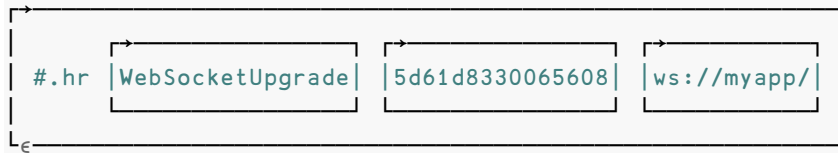
If Type is 'auto', the protocol is handled internally and this event is reported when the connection has already been made. Should the connection fail, a [Section 6.652](#) event will be reported instead.

If Type is 'manual ', a callback function for WebSocketUpgrade is mandatory and is responsible for completing (or denying) the connection. This is achieved by setting the 5th element of the event message (Headers) to indicate an appropriate positive or negative response to the request and returning the entire event message as its result. If a valid response is not generated in this way, the connection will time-out causing a [Section 6.652](#) event. For further information, see <https://tools.ietf.org/html/rfc6455#section-1.3>.

In both cases, the WebSocket ID is subsequently required to send a message by calling the [Section 6.654](#) method or to close the connection using the [Section 6.651](#) method.

Note that several WebSocket connections may be made concurrently.

EXAMPLE



6.656 WeekNumbers Property

Applies To: [Section 6.58](#), [Section 6.143](#)

Description

The WeekNumbers property specifies whether or not a [Section 6.58](#) object displays week numbers.

WeekNumbers is a single number with the value 0 (week numbers are *not* shown) or 1 (week numbers *are* shown); the default is 0.

6.657 Weight Property

Applies To: [Section 6.222](#)

Description

This property specifies the degree of boldness of a font associated with a [Section 6.222](#) object. It is a number in the range 0 to 1000, where 0 represents *very faint* and 1000 represents *very bold*. There is no default; the value of this property reflects the degree of boldness of the font allocated by Windows.

In general, 400 means *normal* and 700 means *bold*.

6.658 WinIniChange Event 133

Applies To: [Section 6.489](#)

Description

If enabled, this event is reported when another application changes relevant registry settings using the standard API calls. The event is reported after the change has taken place and cannot be disabled or inhibited in any way. If your application depends upon registry settings, this event gives you the opportunity of refreshing these parameters if they are changed.

The event message reported as the result of *Dyalog APL Language: Dq*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1]	Object	ref or character vector
[2]	Event	'WinIniChange' or 133

6.659 WordFormat Property

Applies To: [Section 6.488](#)

Description

The WordFormat property is identical to the [Section 6.87](#) property except that it is used to apply formatting to the selected word or words in a [Section 6.488](#) object. If the selection is empty but the insertion point is inside a word, the formatting is applied to the word. See [Section 6.87](#) for further details.

6.660 Wrap Property

Applies To: [Section 6.348](#), [Section 6.459](#), [Section 6.563](#), [Section 6.637](#)

Description

The Wrap property is Boolean and has a default value of 1.

For a [Section 6.348](#) it specifies whether or not long labels (specified by the Items property) may be wrapped or not.

For a [Section 6.459](#) object it determines whether or not the object starts over again when it reaches its upper limit. In particular, if Wrap is 1, the value obtained when you set the Thumb property is given by the expression: `LIMITS[1]+THUMB|LIMITS[2]` where THUMB is the value to which you set the Thumb property and LIMITS is the value of the Limits property.

For a [Section 6.563](#), Wrap determines what happens when the value in the [Section 6.563](#) reaches its upper or lower limit. If Wrap is 1 the [Section 6.563](#) will wrap around to its opposite limit. Otherwise it will stick.

6.661 XRange Property

Applies To

Description

XRange and [Section 6.663](#) together determine a user-defined co-ordinate system. These properties are effective on the object's children which have [Section 6.128](#) set to 'User'.

XRange is a 2-element numeric vector containing the x-coordinate of the top left and bottom right interior corners of the object respectively. See [Section 6.128](#) for further details.

6.662 Yield Property

Applies To: [Section 6.489](#)

Description

This property determines how frequently Dyalog checks the windows message queue. The message queue contains the underlying Windows messages for User interface events, such as mouse clicks, timers, and so on. Dyalog needs to check for the existence of these messages in order to pass them on to any GUI windows that may exist. In addition the windows message queue is used to handle interrupts and to implement the session window.

The value of Yield is an integer expressed in 1/1000's of a second. Its default value is 200. Yield defines the period of time allowed to elapse between the execution of successive lines of APL code before APL yields to Windows by requesting a message from the Windows queue. If Yield is set to zero APL does not explicitly yield.

Note that the value of this property only controls the yield frequency when APL is executing user-defined code. APL may also yield implicitly during *Dyalog APL Language: Dq*, *Dyalog APL Language: Nq*, *Dyalog APL Language: Wc*, $\square$ SR, *Dyalog APL Language: Ws* and *Dyalog APL Language: Wg* and in communicating with Auxiliary processors. Note that setting Yield to 0 (or to a very high value) during the execution of code that does not implicitly yield will effectively de-activate all other applications (including

Program manager) and disable APL interrupts (Ctrl+Break). It should therefore be used with extreme caution.

6.663 YRange Property

Applies To

Description

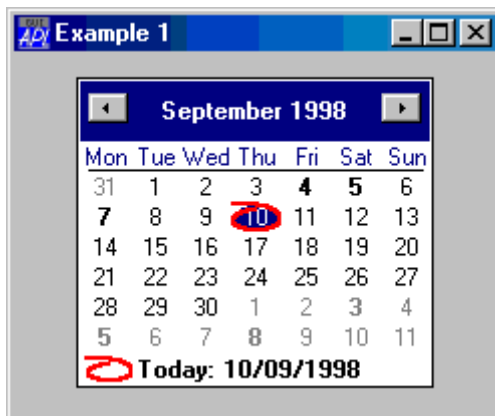
Section 6.661 and YRange together determine a user-defined co-ordinate system. These properties are effective on the object's children which have Section 6.128 set to 'User'.

YRange is a 2-element numeric vector containing the y-coordinate of the top left and bottom right interior corners of the object respectively. See Section 6.128 for further details.

7 Examples

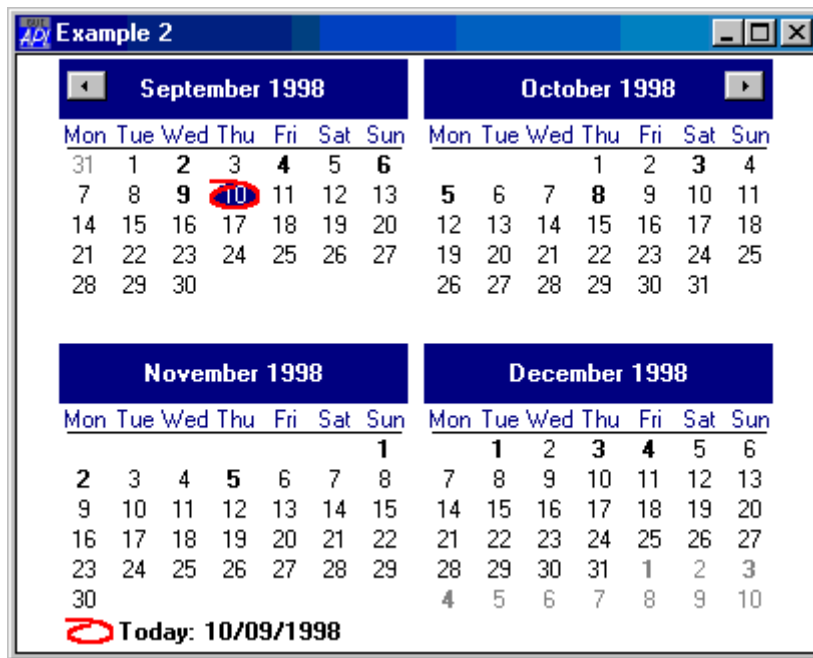
7.1 Calendar Example 1

```
'F'WC'Form' 'Example 1' ('Size' 30 30)  
'F.C'WC'Calendar'
```



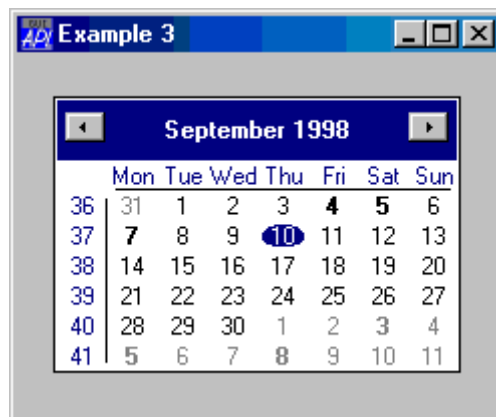
7.2 Calendar Example 2

```
'F'WC'Form' 'Example 2' ('Size' 50 50)  
'F.C'WC'Calendar' ('Size' 100 100)
```



7.3 Calendar Example 3

```
'F'WC'Form' 'Example 3'('Size' 30 30)
'F.C'WC'Calendar'('CircleToday' 0)('HasToday' 0)('WeekNumbers' 1)
```



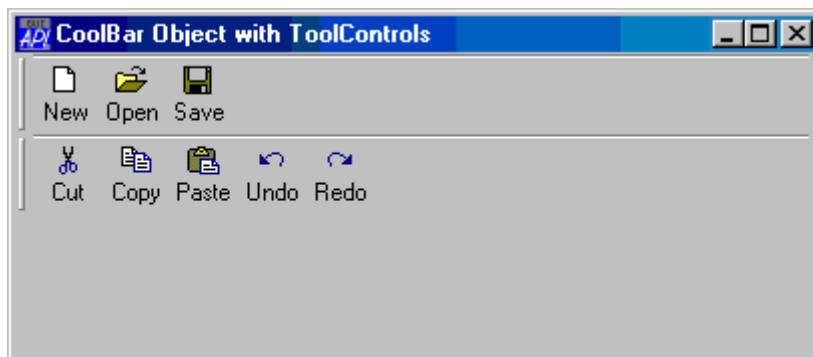
7.4 CoolBar Example 1

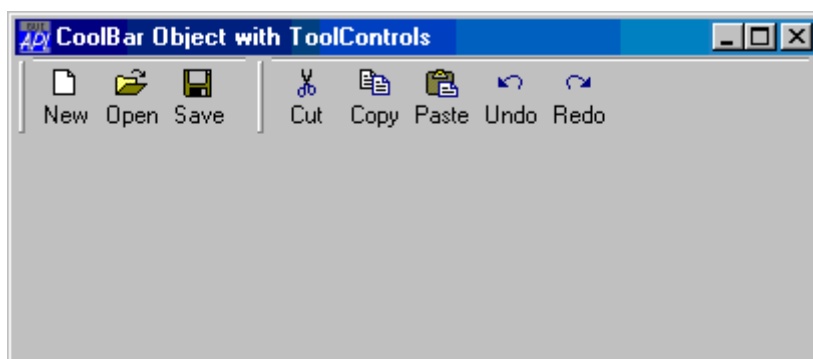
```
'F'WC'Form' 'CoolBar Object with ToolControls'('Size' 25 50)
'F.IL'WC'ImageList'('Masked' 0)('MapCols' 1)
'F.IL.'WC'Bitmap'('ComCtl32' 120)A STD_SMALL

'F.CB'WC'CoolBar'

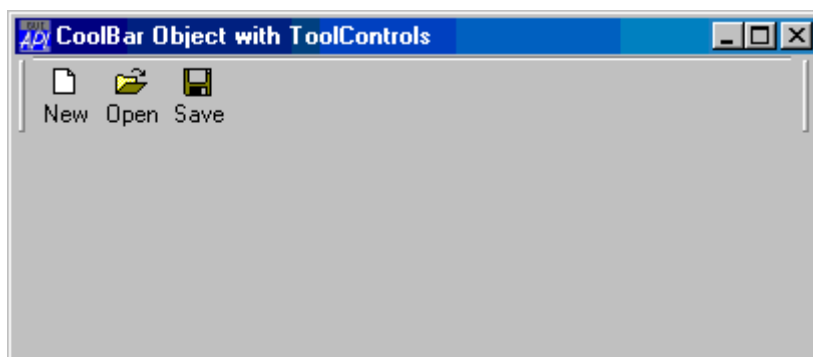
:With 'F.CB.C1'WC'CoolBand'
  'TB'WC'ToolControl'('ImageListObj' '#.F.IL')
  'TB.B1'WC'ToolButton' 'New'('ImageIndex' 7)
  'TB.B2'WC'ToolButton' 'Open'('ImageIndex' 8)
  'TB.B3'WC'ToolButton' 'Save'('ImageIndex' 9)
:EndWith

:With 'F.CB.C2'WC'CoolBand'
  'TB'WC'ToolControl'('ImageListObj' '#.F.IL')
  'TB.B1'WC'ToolButton' 'Cut'('ImageIndex' 1)
  'TB.B2'WC'ToolButton' 'Copy'('ImageIndex' 2)
  'TB.B3'WC'ToolButton' 'Paste'('ImageIndex' 3)
  'TB.B4'WC'ToolButton' 'Undo'('ImageIndex' 4)
  'TB.B5'WC'ToolButton' 'Redo'('ImageIndex' 5)
:EndWith
```

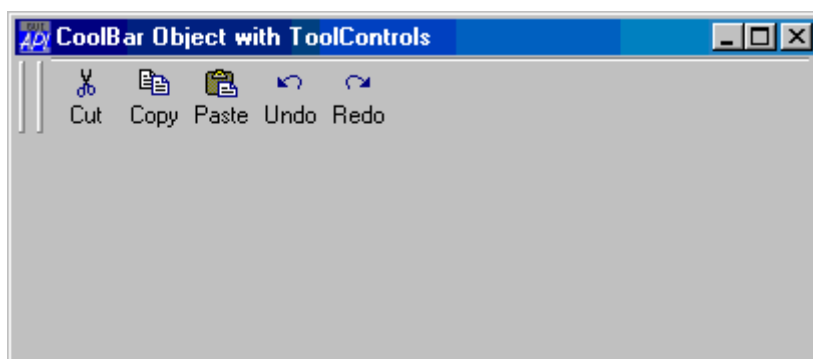




after user has moved band 2 into row 1



after user has maximised band 1



after user has maximised band 2

7.5 CoolBar Example 2

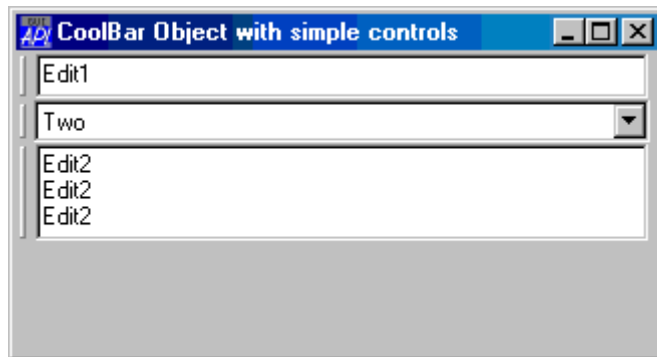
```
'F'□WC'Form' 'CoolBar Object with simple controls'('Size' 25 40)
'F'□WS'Coord' 'Pixel'

'F.CB'□WC'CoolBar'

:With 'F.CB.C1'□WC'CoolBand'
    'E1'□WC'Edit' 'Edit1'
:EndWith

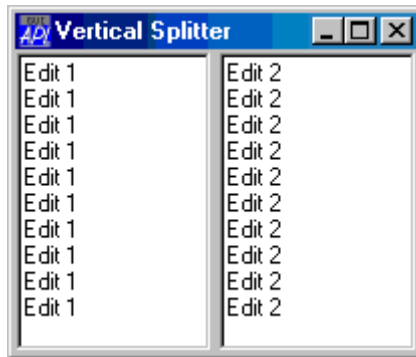
:With 'F.CB.C2'□WC'CoolBand'
    'C1'□WC'Combo'('One' 'Two' 'Three')('SelItems' 0 1 0)
:EndWith

:With 'F.CB.C3'□WC'CoolBand'
    'E2'□WC'Edit'(3 5p'Edit2')('Style' 'Multi')
:EndWith
```



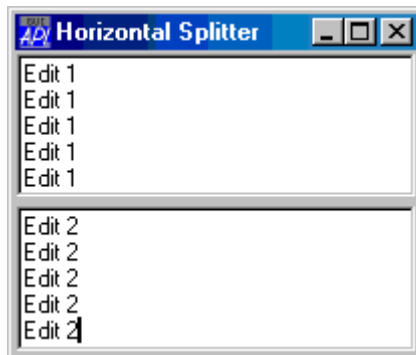
7.6 Splitter Example 1

```
'F'□WC'Form' 'Vertical Splitter'('Size' 25 25)
'F.E1'□WC'Edit'(10 6p'Edit 1')('Style' 'Multi')
'F.E2'□WC'Edit'(10 6p'Edit 2')('Style' 'Multi')
'F.S'□WC'Splitter' 'F.E1' 'F.E2'
```



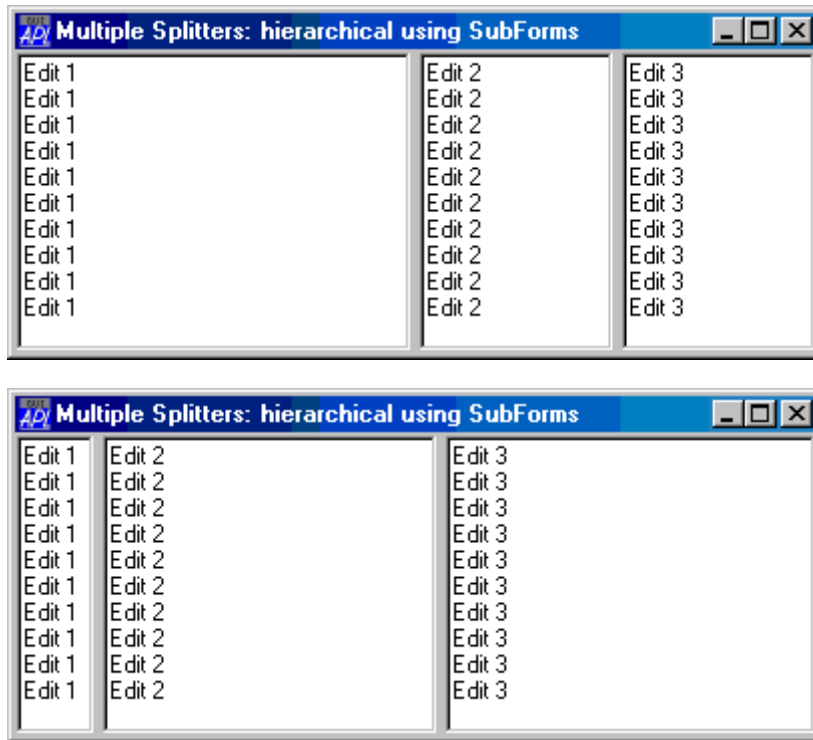
7.7 Splitter Example 2

```
'F'WC'Form' 'Horizontal Splitter'('Size' 25 25)
'F.E1'WC'Edit'(5 6p'Edit 1')('Style' 'Multi')
'F.E2'WC'Edit'(5 6p'Edit 2')('Style' 'Multi')
'F.S'WC'Splitter' 'F.E1' 'F.E2'('Style' 'Horz')
```



7.8 Splitter Example 3

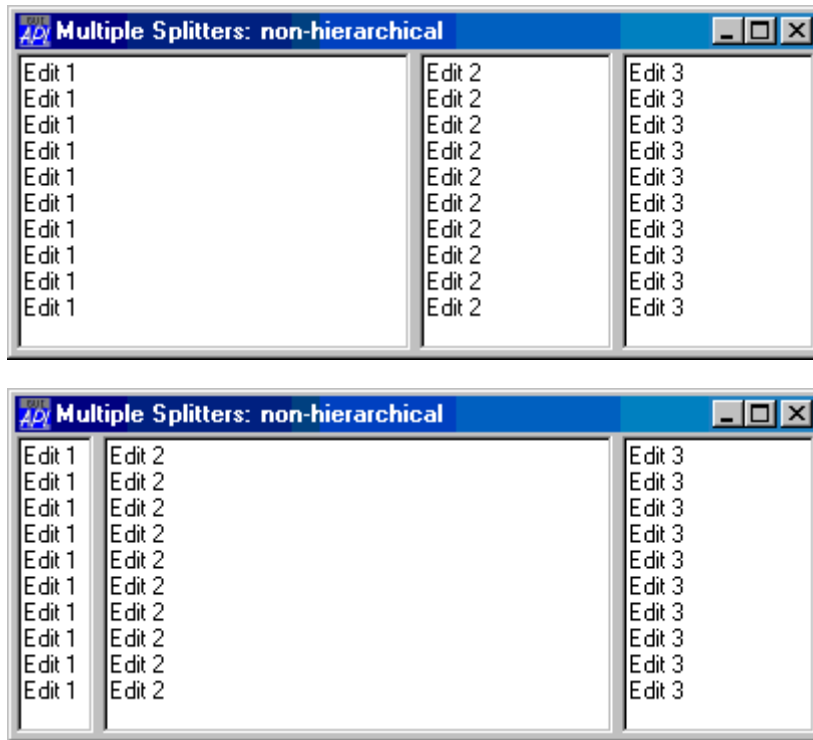
```
'F'WC'Form' 'Multiple Splitters: hierarchical using SubForms'('Size'
25 50)
'F.E1'WC'Edit'(10 6p'Edit 1')('Style' 'Multi')
'F.SF1'WC'SubForm'('EdgeStyle' 'Default')
'F.S1'WC'Splitter' 'F.E1' 'F.SF1'
'F.SF1.E1'WC'Edit'(10 6p'Edit 2')('Style' 'Multi')
'F.SF1.E2'WC'Edit'(10 6p'Edit 3')('Style' 'Multi')
'F.SF1.S1'WC'Splitter' 'F.SF1.E1' 'F.SF1.E2'
```



After dragging the first Splitter to the left.

7.9 Splitter Example 4

```
'F'WC'Form' 'Multiple Splitters: non-hierarchical'('Size' 25 50)
'F.E1'WC'Edit'(10 6p'Edit 1')('Style' 'Multi')
'F.E2'WC'Edit'(10 6p'Edit 2')('Style' 'Multi')
'F.E3'WC'Edit'(10 6p'Edit 3')('Style' 'Multi')
'F.S1'WC'Splitter' 'F.E1'
'F.S2'WC'Splitter' 'F.E2' 'F.E3'
```

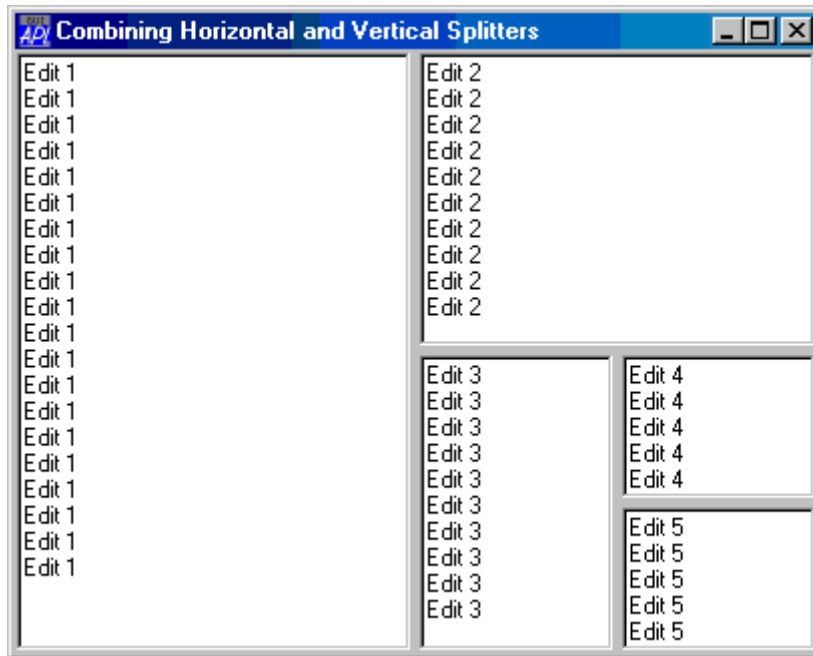



After dragging the first Splitter to the left.

7.10 Splitter Example 5

```
'F'WC'Form' 'Combining Horizontal and Vertical Splitters'
'F.E1'WC'Edit'(20 6p'Edit 1')('Style' 'Multi')
'F.E2'WC'Edit'(10 6p'Edit 2')('Style' 'Multi')
'F.E3'WC'Edit'(10 6p'Edit 3')('Style' 'Multi')
'F.E4'WC'Edit'(5 6p'Edit 4')('Style' 'Multi')
'F.E5'WC'Edit'(5 6p'Edit 5')('Style' 'Multi')

'F.S1'WC'Splitter' 'F.E1'('Style' 'Vert')
'F.S2'WC'Splitter' 'F.E2'('Style' 'Horz')
'F.S3'WC'Splitter' 'F.E3'('Style' 'Vert')
'F.S4'WC'Splitter' 'F.E4' 'F.E5'('Style' 'Horz')
```



7.11 TabControl Example 1

Note that the icons used in this example are provided in the `ws` sub-directory.

```
icodir←(2 □NQ'.' 'GetEnvironment' 'Dyalog'),'\ws\'
```

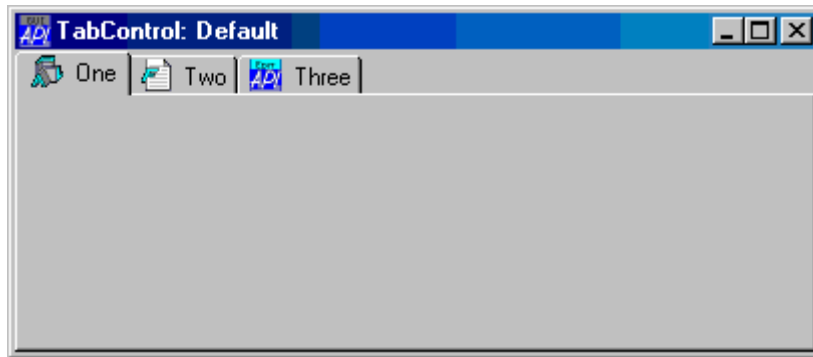
```
'F'□WC'Form' 'TabControl: Default'('Size' 25 50)
'F.TC'□WC'TabControl'
```

```
'F.TC.IL'□WC'ImageList'
'F.TC.IL.'□WC'Icon'(icodir,'aplicon.ico')
'F.TC.IL.'□WC'Icon'(icodir,'funicon.ico')
'F.TC.IL.'□WC'Icon'(icodir,'editicon.ico')
```

```
'F.TC'□WS'ImageListObj' 'F.TC.IL'
```

```
'F.TC.T1'□WC'TabButton' 'One'('ImageIndex' 1)
'F.TC.T2'□WC'TabButton' 'Two'('ImageIndex' 2)
'F.TC.T3'□WC'TabButton' 'Three'('ImageIndex' 3)
```

```
'F.TC.S1'□WC'SubForm'('TabObj' 'F.TC.T1')
'F.TC.S2'□WC'SubForm'('TabObj' 'F.TC.T2')
'F.TC.S3'□WC'SubForm'('TabObj' 'F.TC.T3')
```



7.12 TabControl Example 2

Note that the icons used in this example are provided in the `ws` sub-directory.

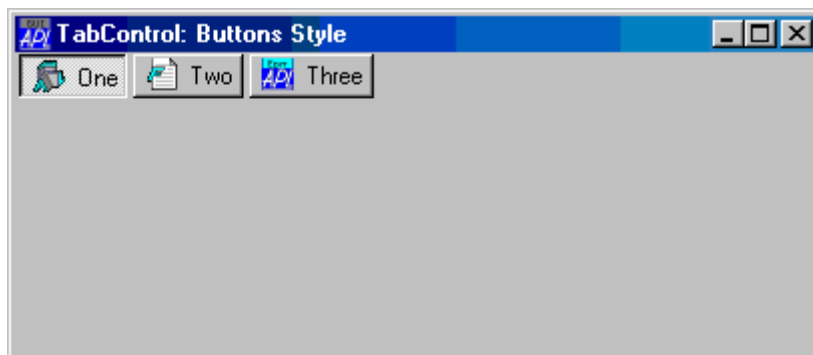
```
icodir←(2 [WC]'.' 'GetEnvironment' 'Dyalog'),'\ws\'

'F'[WC]'Form' 'TabControl: Buttons Style'('Size' 25 50)
'F.TC'[WC]'TabControl'('Style' 'Buttons')

'F.TC.IL'[WC]'ImageList'
'F.TC.IL.'[WC]'Icon'(icodir,'aplicon.ico')
'F.TC.IL.'[WC]'Icon'(icodir,'funicon.ico')
'F.TC.IL.'[WC]'Icon'(icodir,'editicon.ico')

'F.TC'[WC]'ImageListObj' 'F.TC.IL'

'F.TC.T1'[WC]'TabButton' 'One'('ImageIndex' 1)
'F.TC.T2'[WC]'TabButton' 'Two'('ImageIndex' 2)
'F.TC.T3'[WC]'TabButton' 'Three'('ImageIndex' 3)
```



7.13 TabControl Example 3

Note that the icons used in this example are provided in the `ws` sub-directory.

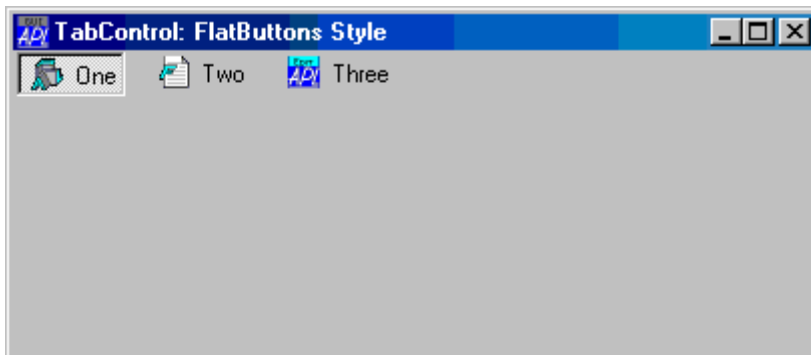
```
icodir←(2 □NQ'.' 'GetEnvironment' 'Dyalog'),'\ws\'

'F'□WC'Form' 'TabControl: FlatButtons Style'('Size' 25 50)
'F.TC'□WC'TabControl'('Style' 'FlatButtons')

'F.TC.IL'□WC'ImageList'
'F.TC.IL.'□WC'Icon'(icodir,'aplicon.ico')
'F.TC.IL.'□WC'Icon'(icodir,'funicon.ico')
'F.TC.IL.'□WC'Icon'(icodir,'editicon.ico')

'F.TC'□WS'ImageListObj' 'F.TC.IL'

'F.TC.T1'□WC'TabButton' 'One'('ImageIndex' 1)
'F.TC.T2'□WC'TabButton' 'Two'('ImageIndex' 2)
'F.TC.T3'□WC'TabButton' 'Three'('ImageIndex' 3)
```



7.14 TabControl Example 3a

Note that the icons used in this example are provided in the `ws` sub-directory.

```
icodir←(2 □NQ'.' 'GetEnvironment' 'Dyalog'),'\ws\'
```

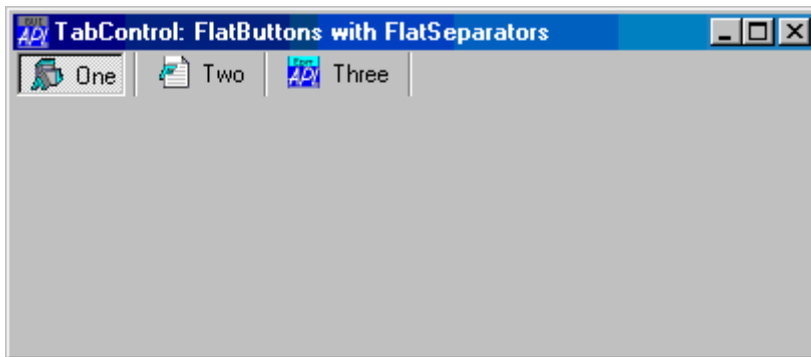
```

'F1'WC'Form' 'TabControl: FlatButtons with FlatSeparators'('Size' 25
50)
'F1.TC'WC'TabControl'('Style' 'FlatButtons')('FlatSeparators' 1)

'F1.TC.IL'WC'ImageList'
'F.TC.IL.'WC'Icon'(icodir,'aplicon.ico')
'F.TC.IL.'WC'Icon'(icodir,'funicon.ico')
'F.TC.IL.'WC'Icon'(icodir,'editicon.ico')
'F1.TC'WS'ImageListObj' 'F1.TC.IL'

'F1.TC.T1'WC'TabButton' 'One'('ImageIndex' 1)
'F1.TC.T2'WC'TabButton' 'Two'('ImageIndex' 2)
'F1.TC.T3'WC'TabButton' 'Three'('ImageIndex' 3)

```



7.15 TabControl Example 4

Note that the icons used in this example are provided in the `ws` sub-directory.

```
icodir←(2 WNQ'.' 'GetEnvironment' 'Dyalog'),'\ws\'
```

```

'F'WC'Form' 'TabControl: Align Bottom'('Size' 25 50)
'F.TC'WC'TabControl'('Align' 'Bottom')

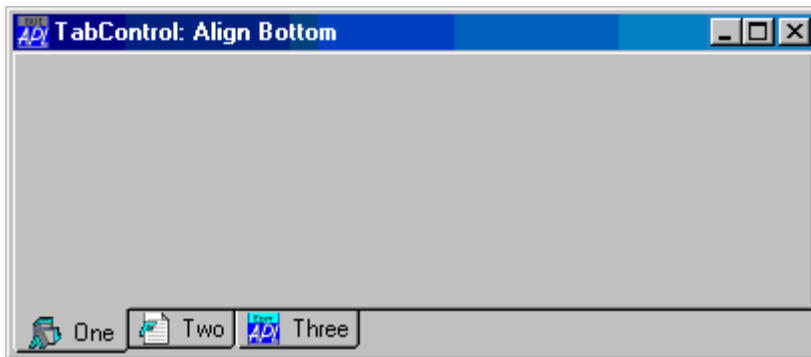
'F.TC.IL'WC'ImageList'
'F.TC.IL.'WC'Icon'(icodir,'aplicon.ico')
'F.TC.IL.'WC'Icon'(icodir,'funicon.ico')
'F.TC.IL.'WC'Icon'(icodir,'editicon.ico')

'F.TC'WS'ImageListObj' 'F.TC.IL'

'F.TC.T1'WC'TabButton' 'One'('ImageIndex' 1)
'F.TC.T2'WC'TabButton' 'Two'('ImageIndex' 2)
'F.TC.T3'WC'TabButton' 'Three'('ImageIndex' 3)

'F.TC.S1'WC'SubForm'('TabObj' 'F.TC.T1')
'F.TC.S2'WC'SubForm'('TabObj' 'F.TC.T2')
'F.TC.S3'WC'SubForm'('TabObj' 'F.TC.T3')

```



7.16 TabControl Example 5

Note that the icons used in this example are provided in the `ws` sub-directory.

```
icodir←(2 WC'.' 'GetEnvironment' 'Dyalog'),'\ws\'
```

```

'F'□WC'Form' 'TabControl: Align Left'('Size' 40 30)
'F.TC'□WC'TabControl'('Align' 'Left')

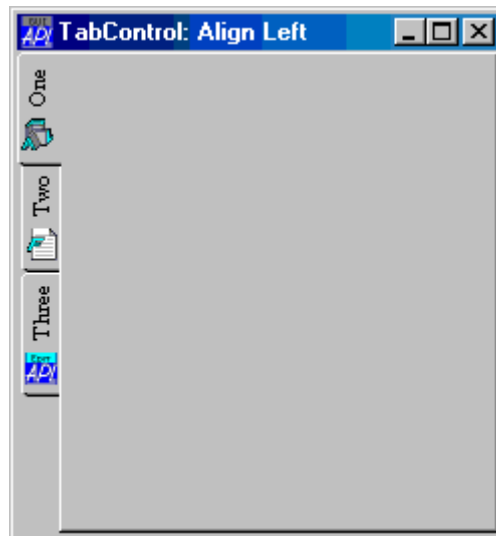
'F.TC.IL'□WC'ImageList'
'F.TC.IL.'□WC'Icon'(icodir,'aplicon.ico')
'F.TC.IL.'□WC'Icon'(icodir,'funicon.ico')
'F.TC.IL.'□WC'Icon'(icodir,'editicon.ico')

'F.TC'□WS'ImageListObj' 'F.TC.IL'

'F.TC.T1'□WC'TabButton' 'One'('ImageIndex' 1)
'F.TC.T2'□WC'TabButton' 'Two'('ImageIndex' 2)
'F.TC.T3'□WC'TabButton' 'Three'('ImageIndex' 3)

'F.TC.S1'□WC'SubForm'('TabObj' 'F.TC.T1')
'F.TC.S2'□WC'SubForm'('TabObj' 'F.TC.T2')
'F.TC.S3'□WC'SubForm'('TabObj' 'F.TC.T3')

```



7.17 TabControl Example 6

Note that the icons used in this example are provided in the `ws` sub-directory.

```
icodir←(2 □NQ'.' 'GetEnvironment' 'Dyalog'),'\ws\'
```

```

'F'WC'Form' 'TabControl: Align Right'('Size' 40 30)
'F.TC'WC'TabControl'('Align' 'Right')

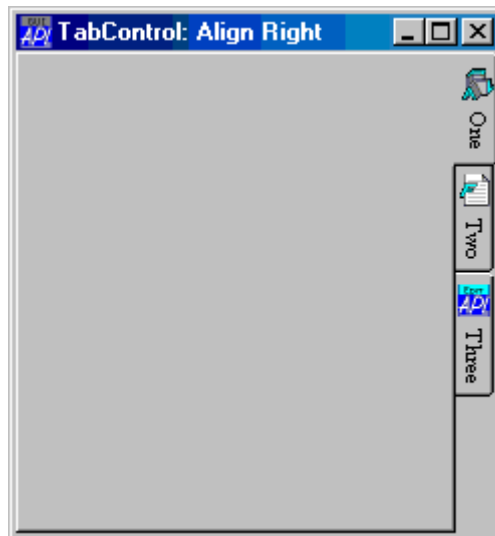
'F.TC.IL'WC'ImageList'
'F.TC.IL.'WC'Icon'(icodir,'aplicon.ico')
'F.TC.IL.'WC'Icon'(icodir,'funicon.ico')
'F.TC.IL.'WC'Icon'(icodir,'editicon.ico')

'F.TC'WS'ImageListObj' 'F.TC.IL'

'F.TC.T1'WC'TabButton' 'One'('ImageIndex' 1)
'F.TC.T2'WC'TabButton' 'Two'('ImageIndex' 2)
'F.TC.T3'WC'TabButton' 'Three'('ImageIndex' 3)

'F.TC.S1'WC'SubForm'('TabObj' 'F.TC.T1')
'F.TC.S2'WC'SubForm'('TabObj' 'F.TC.T2')
'F.TC.S3'WC'SubForm'('TabObj' 'F.TC.T3')

```



7.18 TabControl Example 7

Note that the icons used in this example are provided in the `ws` sub-directory.

```
icodir←(2 WNQ'.' 'GetEnvironment' 'Dyalog'),'\ws\'
```



```

'F'□WC'Form' 'TabControl: Default Scrolling'('Size' 25 40)
'F.TC'□WC'TabControl'

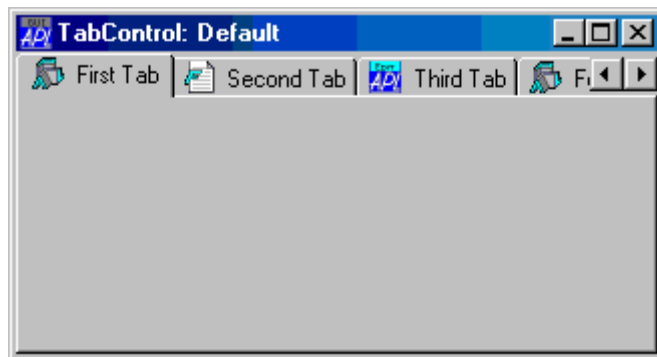
'F.TC.IL'□WC'ImageList'
'F.TC.IL.'□WC'Icon'(icodir,'aplicon.ico')
'F.TC.IL.'□WC'Icon'(icodir,'funicon.ico')
'F.TC.IL.'□WC'Icon'(icodir,'editicon.ico')

'F.TC'□WS'ImageListObj' 'F.TC.IL'

'F.TC.T1'□WC'TabButton' 'First Tab'('ImageIndex' 1)
'F.TC.T2'□WC'TabButton' 'Second Tab'('ImageIndex' 2)
'F.TC.T3'□WC'TabButton' 'Third Tab'('ImageIndex' 3)
'F.TC.T4'□WC'TabButton' 'Fourth Tab'('ImageIndex' 1)

'F.TC.S1'□WC'SubForm'('TabObj' 'F.TC.T1')
'F.TC.S2'□WC'SubForm'('TabObj' 'F.TC.T2')
'F.TC.S3'□WC'SubForm'('TabObj' 'F.TC.T3')
'F.TC.S4'□WC'SubForm'('TabObj' 'F.TC.T4')

```



7.19 TabControl Example 8

Note that the icons used in this example are provided in the `ws` sub-directory.

```
icodir←(2 □NQ'.' 'GetEnvironment' 'Dyalog'),'\ws\'
```

```

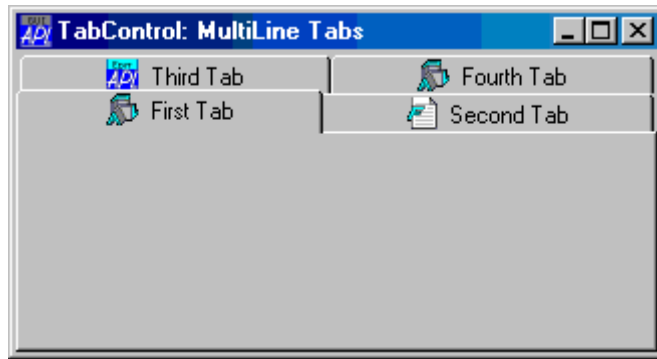
'F'WC'Form' 'TabControl: MultiLine Tabs'('Size' 25 40)
'F.TC'WC'TabControl'('MultiLine' 1)

'F.TC.IL'WC'ImageList'
'F.TC.IL.'WC'Icon'(icodir,'aplicon.ico')
'F.TC.IL.'WC'Icon'(icodir,'funicon.ico')
'F.TC.IL.'WC'Icon'(icodir,'editicon.ico')

'F.TC'WS'ImageListObj' 'F.TC.IL'

'F.TC.T1'WC'TabButton' 'First Tab'('ImageIndex' 1)
'F.TC.T2'WC'TabButton' 'Second Tab'('ImageIndex' 2)
'F.TC.T3'WC'TabButton' 'Third Tab'('ImageIndex' 3)
'F.TC.T4'WC'TabButton' 'Fourth Tab'('ImageIndex' 1)

```



7.20 TabControl Example 9

Note that the icons used in this example are provided in the `ws` sub-directory.

```
icodir←(2 WC'.' 'GetEnvironment' 'Dyalog'),'\ws\'
```

```

'F'WC'Form' 'TabControl: MultiLine Buttons'('Size' 25 40)
'F.TC'WC'TabControl'('Style' 'Buttons')('MultiLine' 1))

'F.TC.IL'WC'ImageList'
'F.TC.IL.'WC'Icon'(icodir,'aplicon.ico')
'F.TC.IL.'WC'Icon'(icodir,'funicon.ico')
'F.TC.IL.'WC'Icon'(icodir,'editicon.ico')
'F.TC'WS'ImageListObj' 'F.TC.IL'

'F.TC.T1'WC'TabButton' 'First Tab'('ImageIndex' 1)
'F.TC.T2'WC'TabButton' 'Second Tab'('ImageIndex' 2)
'F.TC.T3'WC'TabButton' 'Third Tab'('ImageIndex' 3)
'F.TC.T4'WC'TabButton' 'Fourth Tab'('ImageIndex' 1)

```



7.21 TabControl Example 10

Note that the icons used in this example are provided in the `ws` sub-directory.

```
icodir←(2 WNQ'.' 'GetEnvironment' 'Dyalog'),'\ws\'
```

```

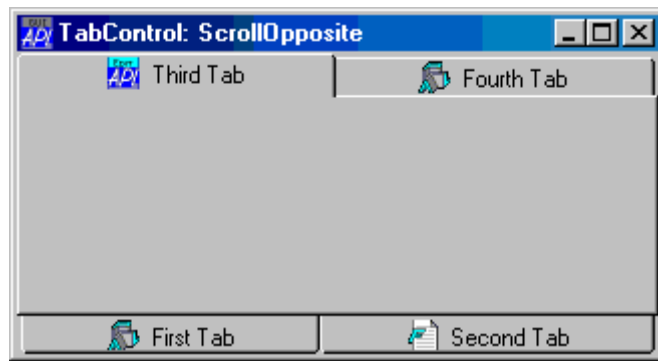
'F'WC'Form' 'TabControl: ScrollOpposite'('Size' 25 40)
'F.TC'WC'TabControl'('ScrollOpposite' 1)

'F.TC.IL'WC'ImageList'
'F.TC.IL.'WC'Icon'(icodir,'aplicon.ico')
'F.TC.IL.'WC'Icon'(icodir,'funicon.ico')
'F.TC.IL.'WC'Icon'(icodir,'editicon.ico')

'F.TC'WS'ImageListObj' 'F.TC.IL'

'F.TC.T1'WC'TabButton' 'First Tab'('ImageIndex' 1)
'F.TC.T2'WC'TabButton' 'Second Tab'('ImageIndex' 2)
'F.TC.T3'WC'TabButton' 'Third Tab'('ImageIndex' 3)
'F.TC.T4'WC'TabButton' 'Fourth Tab'('ImageIndex' 1)

```



7.22 TabControl Example 11

Note that the icons used in this example are provided in the `ws` sub-directory.

```
icodir←(2 WNQ'.' 'GetEnvironment' 'Dyalog'),'\ws\'
```

```

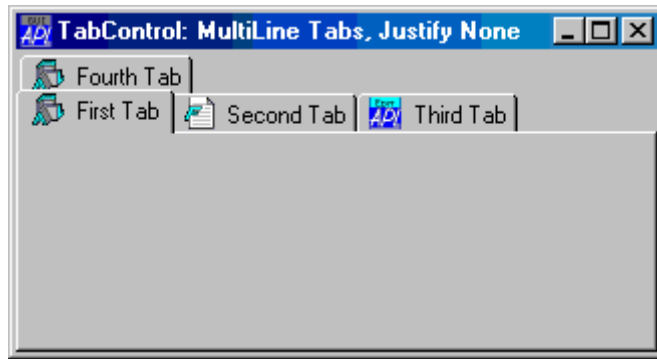
'F'WC'Form' 'TabControl: MultiLine Tabs, Justify None'('Size' 25 40)
'F.TC'WC'TabControl'('MultiLine' 1)('Justify ' 'None')

'F.TC.IL'WC'ImageList'
'F.TC.IL.'WC'Icon'(icodir,'aplicon.ico')
'F.TC.IL.'WC'Icon'(icodir,'funicon.ico')
'F.TC.IL.'WC'Icon'(icodir,'editicon.ico')

'F.TC'WS'ImageListObj' 'F.TC.IL'

'F.TC.T1'WC'TabButton' 'First Tab'('ImageIndex' 1)
'F.TC.T2'WC'TabButton' 'Second Tab'('ImageIndex' 2)
'F.TC.T3'WC'TabButton' 'Third Tab'('ImageIndex' 3)
'F.TC.T4'WC'TabButton' 'Fourth Tab'('ImageIndex' 1)

```



7.23 TabControl Example 12

Note that the icons used in this example are provided in the `ws` sub-directory.

```
icodir←(2 WNQ'.' 'GetEnvironment' 'Dyalog'),'\ws\'
```

```

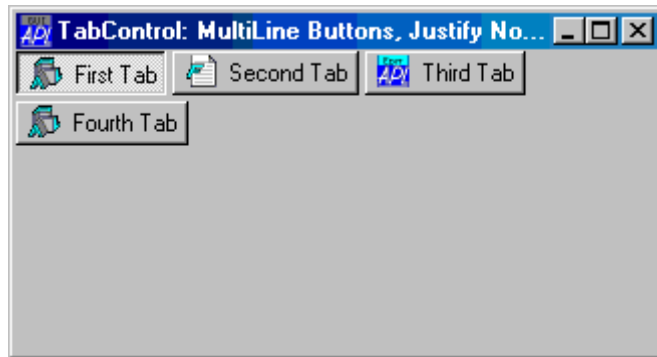
'F'WC'Form' 'TabControl: MultiLine Buttons, Justify None'('Size' 25 40)
'F.TC'WC'TabControl'('Style ' 'Buttons') ('MultiLine' 1)('Justify ' 'No
ne')

'F.TC.IL'WC'ImageList'
'F.TC.IL.'WC'Icon'(icodir,'aplicon.ico')
'F.TC.IL.'WC'Icon'(icodir,'funicon.ico')
'F.TC.IL.'WC'Icon'(icodir,'editicon.ico')

'F.TC'WS'ImageListObj' 'F.TC.IL'

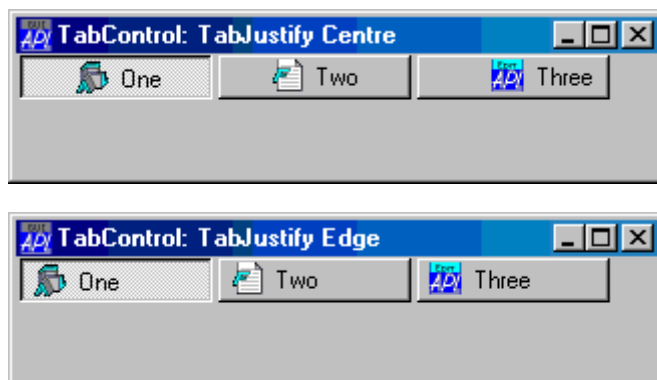
'F.TC.T1'WC'TabButton' 'First Tab'('ImageIndex' 1)
'F.TC.T2'WC'TabButton' 'Second Tab'('ImageIndex' 2)
'F.TC.T3'WC'TabButton' 'Third Tab'('ImageIndex' 3)
'F.TC.T4'WC'TabButton' 'Fourth Tab'('ImageIndex' 1)

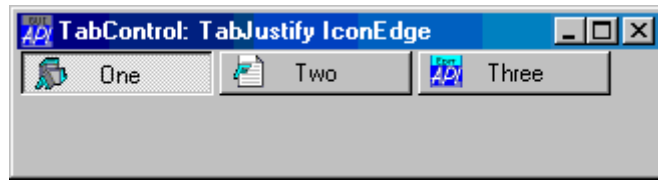
```



7.24 TabControl Example 13

The TabJustify Property.





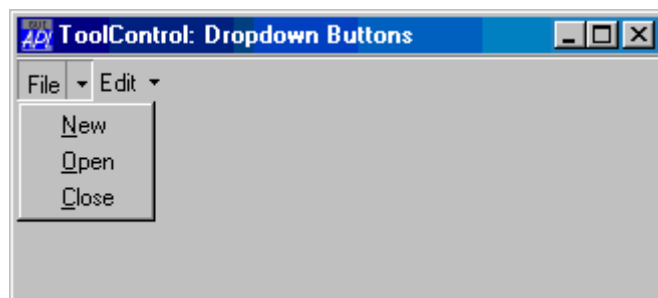
7.25 ToolButton Example 1

```
'F'WC'Form' 'ToolControl: Dropdown Buttons'('Size' 20 40)
'F.TB'WC'ToolControl'('ShowDropDown' 1)

:With 'F.FMENU'WC'Menu' A Popup File menu
  'NEW'WC'MenuItem' '&New'
  'OPEN'WC'MenuItem' '&Open'
  'CLOSE'WC'MenuItem' '&Close'
:EndWith

:With 'F.EMENU'WC'Menu' A Popup File menu
  'CUT'WC'MenuItem' 'Cu&t'
  'COPY'WC'MenuItem' '&Copy'
  'PASTE'WC'MenuItem' '&Paste'
:EndWith

'F.TB.B1'WC'ToolButton' 'File'('Style' 'DropDown')('Popup' 'F.FMENU')
'F.TB.B2'WC'ToolButton' 'Edit'('Style' 'DropDown')('Popup' 'F.EMENU')
```

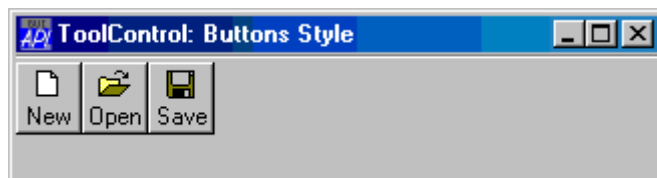
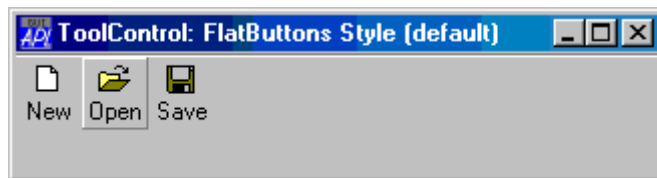


7.26 ToolControl Example 1

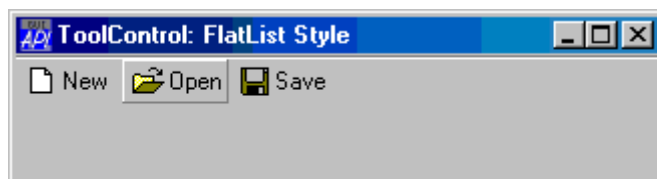
```
'F'WC'Form' 'ToolControl: FlatButtons Style (default)'('Size' 10 40)
'F.TB'WC'ToolControl'

'F.TB.IL'WC'ImageList'('Masked' 0)
'F.TB.IL.'WC'Bitmap'('Comctl32' 120)
'F.TB'WS'ImageListObj' 'F.TB.IL'

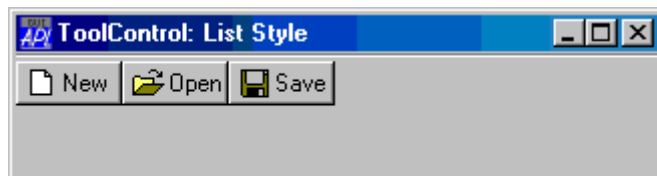
'F.TB.B1'WC'ToolButton' 'New'('ImageIndex' 7)
'F.TB.B2'WC'ToolButton' 'Open'('ImageIndex' 8)
'F.TB.B3'WC'ToolButton' 'Save'('ImageIndex' 9)
```



```
'F.TB'WC'ToolControl'('Style' 'Buttons')
```



```
'F.TB'WC'ToolControl'('Style' 'FlatList')
```



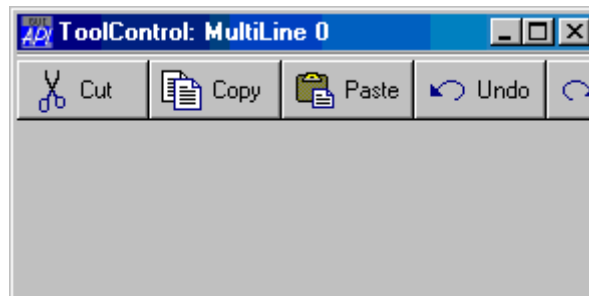
```
'F.TB'WC'ToolControl'('Style' 'List')
```


7.27 ToolControl Example 2

```
'F'WC'Form' 'ToolControl: MultiLine 0'('Size' 20 36)
'F.TB'WC'ToolControl'('Style' 'List')

'F.TB.IL'WC'ImageList'('Masked' 0)('Size' 24 24)
'F.TB.IL.'WC'Bitmap'('ComCtl32' 121)A STD_LARGE
'F.TB'WS'ImageListObj' 'F.TB.IL'

'F.TB.B1'WC'ToolButton' 'Cut'('ImageIndex' 1)
'F.TB.B2'WC'ToolButton' 'Copy'('ImageIndex' 2)
'F.TB.B3'WC'ToolButton' 'Paste'('ImageIndex' 3)
'F.TB.B4'WC'ToolButton' 'Undo'('ImageIndex' 4)
'F.TB.B5'WC'ToolButton' 'Redo'('ImageIndex' 5)
'F.TB.B6'WC'ToolButton' 'Delete'('ImageIndex' 6)
```



```
'F.TB'WC'ToolControl'('MultiLine' 1)('Style' 'List')
```

7.28 ToolControl Example 3

```
'F'WC'Form' 'ToolControl: Transparent 1'('Size' 10 40)
'F.BM'WC'Bitmap' 'C:\WINDOWS\WINLOGO'
'F'WS'Picture' 'F.BM' 1

'F.TB'WC'ToolControl'('Transparent' 1)('Style' 'FlatList')
'F.TB.IL'WC'ImageList'('Masked' 0)('Size' 24 24)
'F.TB.IL.'WC'Bitmap'('ComCtl32' 121)A STD_LARGE
'F.TB'WS'ImageListObj' 'F.TB.IL'

'F.TB.B1'WC'ToolButton' 'New'('ImageIndex' 7)
'F.TB.B2'WC'ToolButton' 'Open'('ImageIndex' 8)
'F.TB.B3'WC'ToolButton' 'Save'('ImageIndex' 9)
```



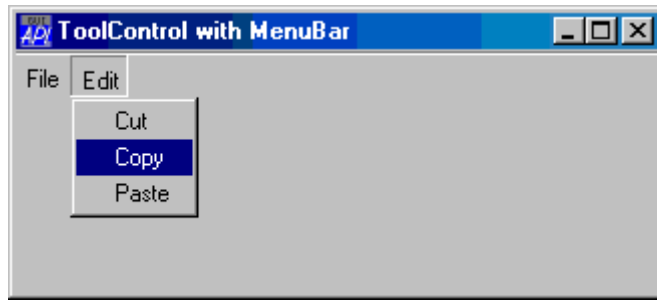
7.29 ToolControl Example 4

```
'F'WC'Form' 'ToolControl with MenuBar'('Size' 20 40)
'F.TB'WC'ToolControl'

:With 'F.TB.MB'WC'MenuBar'
  :With 'File'WC'Menu' 'File'
    'New'WC'MenuItem' 'New'
    'Open'WC'MenuItem' 'Open'
    'Close'WC'MenuItem' 'Close'
  :EndWith

  :With 'Edit'WC'Menu' 'Edit'
    'Cut'WC'MenuItem' 'Cut'
    'Copy'WC'MenuItem' 'Copy'
    'Paste'WC'MenuItem' 'Paste'
  :EndWith

:EndWith
```



8 Miscellaneous

8.1 Class Identifier (CLSID)

The CLSID is a unique identifier that, when displayed, appears as a string of hexadecimal digits wrapped in braces; for example "{3B7C8860-D78F-101B-B9B5-04021C009402}".

8.2 COM data types

The table below shows the correspondence between COM data types and APL arrays.

OLE DataType	APL array
VT_BOOL	numeric scalar
VT_I1	numeric scalar
VT_I2	numeric scalar
VT_I4	numeric scalar
VT_R4	numeric scalar
VT_R8	numeric scalar
VT_BSTR	character vector
VT_CY	2-element numeric vector
VT_DATE	6 element numeric vector
VT_VARIANT	any array
VT_SAFEARRAY	any array (VT_ARRAY OF VT_VARIANT)
VT_DISPATCH	□OR of a namespace
VT_COLOR	3-element RGB

APL vectors may be described by pre-fixing the data type string with 'VT_ARRAY OF '. For example 'VT_ARRAY OF BSTR' specifies a vector of character vectors.

If the APL array is the `⌈0R` of a namespace, its data type should be specified as `'VT_DISPATCH'`.

8.3 DISPID (Dispatch ID)

COM objects created by Dyalog (OLEServer and ActiveXControl objects) export their members (methods, properties and events) using the standard *IDispatch* interface.

Using this interface, a client application may discover the names and parameters of the members supported by an object at run-time, and then access them by name. Alternatively, a client application may compile references to the object's members in advance using their *Dispatch IDs* or *DISPIDs*.

Prior to Version 14.1, Dyalog assigned all DISPIDs automatically¹, making it impractical for them to be compiled into client applications.

From Version 14.1 onwards, the `SetFnInfo`, `SetPropertyInfo` and `SetEventInfo` methods allow the Dyalog programmer to assign DISPIDs so that they may be used directly by client applications. The specified DISPID must be a non-zero integer. The special value `-1` causes Dyalog to assign the DISPID automatically as before.

Note

Each of the DISPIDs exported by a COM object must be unique. Furthermore, the behaviour of a COM object with non-unique DISPIDs is undefined. Non-unique DISPIDs may prevent the COM object from being registered (with or without generating an error) or may cause a run-time failure. If Dyalog assigns all the DISPIDs of an object, they will be unique. If you choose to allocate your own DISPIDs to **any** of the members of a Dyalog COM object, the responsibility to ensure that they are **all** unique is yours. In this case, Dyalog does not guarantee nor check for uniqueness.

8.4 Globally Unique Identifier (GUID)

A Globally Unique Identifier (GUID) is a 128-bit number that is generated by a special algorithm and is virtually guaranteed to be unique in the world. That is to say that every GUID generated by this algorithm is different from any other.

Dyalog APL properties that represent GUIDs are read-only character vectors that represent the GUID as a string of hexadecimal digits wrapped in braces.

¹ An automatically assigned DISPID is its index into the list of the names of the object's members in alphabetic order, and may therefore change when this list is altered in any way.

For example "{3B7C8860-D78F-101B-B9B5-04021C009402}".

8.5 International Day Number

The values of dates used by the properties, events and methods of the Calendar object are specified using IDNs (International Day Numbers).

An IDN is an integer that represents a date as the number of days starting at Jan 1st 1900, that is, Jan 01 1900 is IDN 1.

Note that the value of IDNs calculated by Dyalog APL are one less than IDNs calculated by Microsoft Excel which incorrectly assumes that 1900 was a leap year.

8.6 Native Look and Feel

Native Look and Feel is a Dyalog option that affects the appearance of the controls provided by the Dyalog GUI Interface and those used by the Dyalog Session. It is implemented by the **XPLookAndFeel** parameter.

Most of the Dyalog controls (with the notable exception of the Dyalog Grid) are standard Windows user-interface components provided by the Windows Common Controls library `comctl32.dll`. Successive versions of Windows have introduced new versions of the Windows Common Control Library which typically provide additional features as well as certain differences in appearance. However, each version of Windows continues to support older versions of the Common Control Library as well as the latest one. The decision as to which is loaded is made at run-time.

A second factor that affects the appearance of user-interface controls is the application of *Visual Styles*² and Themes. These features enable users to tailor the UI to accommodate their individual needs and preferences. From Windows 8 onwards, the default appearance of certain Common Controls is overridden by the Visual Styles in use. However, this applies only if *Native Look and Feel* is enabled.

If *Native Look and Feel* is enabled, Windows loads the latest version of `comctl32.dll` (and potentially other Windows dlls) that is appropriate for the version of Windows in use. If *Native Look and Feel* is disabled, an earlier version may be loaded. The specific version that is loaded is not determined by Dyalog, but by Windows.

The Dyalog for Microsoft Windows Object Reference Guide identifies which features require Native Look and Feel to be enabled. It documents the typical appearance of

² See msdn.microsoft.com for details.

controls with Native Look and Feel disabled but does not specify how the appearance of controls is affected by enabling Native Look and Feel, which is in any case affected by the Visual Styles selected by the user.

Dyalog Session

During development, both the Dyalog Session and the Dyalog APL GUI will display native style buttons, combo boxes, and other GUI components if *Native Look and Feel* is enabled. The option is provided in the *General* tab of the *Configuration* dialog.

Applications

By default *Native Look and Feel* is disabled in end-user applications. There are two ways to enable it.

If you use the *File/Export...* menu item on the Session MenuBar to create a bound executable, an OLE Server (in-process or out-of-process), an ActiveX Control or a .NET Assembly, check the option box labelled *Enable Native Look and Feel* in the *create bound file* dialog box. See *Installation/Configuration: Creating Executables*.

If not, set the **XPLookandFeel** parameter to 1, when you run the program. For example:

```
dyalogrt.exe XPLookAndFeel=1 myws.dws
```

8.7 Port Number

A port number identifies the service provided by a particular server. It is an integer number in the range 0-65536,

Port numbers in the range 0-1023 are reserved for "well-known" services (for example, FTP and HTTP).

Port number 1024 is reserved for IANA.

Port numbers 1025-5000 are typically assigned for user-defined services.

8.8 Stream Sockets

A Stream socket is a connection-based transport that is analogous to a telephone service. A Stream socket handles error correction, guarantees delivery, and preserves data sequence. This means that if you send two messages to a recipient, the messages are sure to arrive and in the sequence that you sent them. However, individual

messages may be broken up into several packets (or accumulated into one), and there is no predetermined protocol to identify message boundaries. This means that Stream-based applications must implement some kind of message protocol that both ends of a connection understand and adhere to.

8.9 Twips

There are 1,440 Twips to one inch.

8.10 User Datagram Protocol (UDP)

User Datagram Protocol (UDP) is a connection-less transport mechanism that is somewhat similar to a postal service. It permits a sending application to transmit a message or messages to a recipient. It neither guarantees delivery nor acknowledgment, nor does it preserve the sequence of messages. Messages are also limited to fit into a single packet which is typically no more than 1500 bytes in size. However, a UDP message will be delivered in its entirety.

You may wonder why anybody would use a service that does not guarantee delivery. The answer is that although UDP is technically an unreliable service, it is perfectly possible to implement reliable applications on top of it by building in acknowledgments, time-outs and re-transmissions.