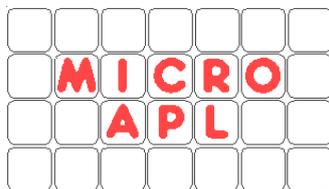




APLX

APLX System Classes and User-Interface Programming

Version 5.0



Copyright © 1985-2009 MicroAPL Ltd. All rights reserved worldwide.

APLX, APL.68000 and MicroAPL are trademarks of MicroAPL Ltd. All other trademarks acknowledged.

APLX is a proprietary product of MicroAPL Ltd, and its use is subject to the license agreement in force. Unauthorized copying or use of APLX is illegal.

MicroAPL Ltd makes no warranties in respect of the suitability of APLX for any particular purpose, and accepts no liability for any loss arising out of the use of APLX or arising from the information contained in this manual.

MicroAPL welcomes your comments and suggestions.
Please visit our website: <http://www.microapl.co.uk/apl>

Version 5.0 July 2009

Contents

Section 1. Introduction to APLX System Classes	15
Introduction to System Classes	17
System Classes by category	24
Positioning controls in windows	27
Events and Callbacks	30
Modal Dialogs	32
Section 2. List of APLX System Classes	35
APL (Child task) Object	37
Arc	38
Bevel	40
Browser	42
Button	45
Chart	47
Check	49
ChooseColor	51
ChooseDir	53
ChooseFont	54
Combo	56
Dialog	58
Document	59
Edit	60
Form	62
Frame	66
GetMail	68
Grid	74
Icon	84
HTTPClient	85
Image	90
ImageList	97
Label	102
Line	104
List	106
Menu	108
Movie	112
MsgBox	114
OLEContainer	116
OpenFile	120
Page	123
Picture	125
Printer	128
Progress	131
Radio	133
Rectangle	135
RichEdit	137
RoundRect	141

SaveFile	143
Scroll	146
Selector	148
SendMail	149
Series	154
Socket	155
Spinner	160
Splitter	162
System	164
Timer	168
ToolButton	169
Trackbar	173
Tree	176
Section 3. System Classes: Properties	181
The 'action' property	183
The 'activecell' property	183
The 'align' property	184
The 'allowselection' property	184
The 'anchors' property	185
The 'angle' property	185
The 'aplkeyboard' property	186
The 'aquaadjust' property	186
The 'attachments' property	187
The 'autoactivate' property	187
The 'autodraw' property	188
The 'autoeditstart' property	188
The 'axiswidth' property	189
The 'background' property	189
The 'barwidth' property	189
The 'bcc' property	190
The 'bitmap' property	190
The 'bitmapsizes' property	191
The 'body' property	191
The 'border' property	192
The 'borderstyle' property	193
The 'canundo' property	193
The 'caption' property	193
The 'cc' property	194
The 'children' property	195
The 'class' property	195
The 'classes' property	195
The 'closevalues' property	196
The 'col' property	196
The 'color' property	196
The 'coloraxis' property	198
The 'colorback' property	198
The 'colorgrid' property	199
The 'colorhead' property	199
The 'colorlegend' property	200
The 'colormarker' property	200
The 'colornote' property	201

The 'colortext' property	201
The 'colortitle' property	202
The 'colour' property	202
The 'cols' property	202
The 'colsize' property	203
The 'contents' property	203
The 'conversionerrorvalue' property	204
The 'cookie' property	205
The 'copies' property	205
The 'count' property	205
The 'custom' property	206
The 'data' property	206
The 'date' property	207
The 'def' property	207
The 'default' property	207
The 'deleteonread' property	208
The 'directory' property	208
The 'docstate' property	209
The 'doublebuffered' property	209
The 'dragsource' property	210
The 'droptarget' property	210
The 'enabled' property	210
The 'eventmask' property	211
The 'events' property	211
The 'extent' property	211
The 'family' property	212
The 'file' property	213
The 'filled' property	215
The 'fillmarker' property	215
The 'fillpattern' property	215
The 'filter' property	216
The 'filterindex' property	216
The 'firstvisible' property	217
The 'font' property	217
The 'fontaxis' property	218
The 'fontlegend' property	219
The 'fontnote' property	219
The 'fonts' property	220
The 'fontstyle' property	220
The 'fonttitle' property	221
The 'format' property	221
The 'formats' property	222
The 'from' property	223
The 'gridlines' property	223
The 'gridwidth' property	224
The 'group' property	224
The 'handle' property	225
The 'hdc' property	225
The 'headcols' property	226
The 'header' property	226
The 'headrows' property	227

The 'highlightbold' property	227
The 'highlightcut' property	227
The 'highlightdrop' property	228
The 'highlightselect' property	228
The 'highvalues' property	228
The 'host' property	229
The 'html' property	230
The 'icon' property	230
The 'id' property	231
The 'imagealloc' property	231
The 'imagecount' property	231
The 'imageindex' property	232
The 'imagelist' property	232
The 'imagelistuser' property	233
The 'imagenames' property	233
The 'imagesize' property	234
The 'increment' property	235
The 'indent' property	235
The 'interval' property	235
The 'labeledithwnd' property	236
The 'limit' property	236
The 'linecount' property	236
The 'lineheight' property	237
The 'linetype' property	237
The 'linewidth' property	238
The 'list' property	238
The 'lowvalues' property	239
The 'margin' property	239
The 'marker' property	240
The 'maskcolor' property	240
The 'maxsize' property	241
The 'menuimagelist' property	241
The 'messages' property	242
The 'methods' property	242
The 'minsize' property	243
The 'modified' property	243
The 'monochrome' property	243
The 'movieref' property	244
The 'name' property	244
The 'note' property	245
The 'offline' property	245
The 'oleclasses' property	246
The 'oledotypes' property	247
The 'opened' property	248
The 'openvalues' property	249
The 'order' property	249
The 'orientation' property	250
The 'overlays' property	250
The 'page' property	250
The 'password' property	251
The 'path' property	251

The 'pen' property	251
The 'pensize' property	252
The 'picture' property	252
The 'pitch' property	253
The 'placelegend' property	254
The 'placenote' property	254
The 'placetitle' property	254
The 'playing' property	255
The 'pointer' property	255
The 'port' property	256
The 'position' property	256
The 'printername' property	257
The 'printers' property	257
The 'progid' property	258
The 'properties' property	258
The 'protocol' property	259
The 'proxy' property	259
The 'quality' property	259
The 'range' property	260
The 'referrer' property	260
The 'replyto' property	261
The 'rounding' property	261
The 'row' property	261
The 'rows' property	262
The 'rowsize' property	262
The 'rtf' property	263
The 'scale' property	263
The 'searchstring' property	264
The 'selalign' property	265
The 'selbullet' property	265
The 'selcolor' property	265
The 'selection' property	266
The 'self' property	266
The 'selfont' property	267
The 'selindents' property	267
The 'selrtf' property	267
The 'selstyle' property	268
The 'seltabs' property	268
The 'seltext' property	269
The 'separator' property	269
The 'serverreply' property	269
The 'shortcut' property	270
The 'size' property	270
The 'sizemode' property	271
The 'sliderlen' property	272
The 'sliderwhere' property	272
The 'sourceformats' property	272
The 'state' property	273
The 'status' property	273
The 'style' property	274
The 'subject' property	274

The 'subtitle' property	275
The 'svg' property	275
The 'tabgroup' property	275
The 'tabparent' property	276
The 'tabrows' property	276
The 'tabstop' property	276
The 'targetformats' property	277
The 'taskid' property	277
The 'text' property	278
The 'textalign' property	279
The 'tickinterval' property	279
The 'tickpos' property	280
The 'ticks' property	280
The 'tie' property	280
The 'timeout' property	281
The 'title' property	281
The 'to' property	281
The 'tooltip' property	282
The 'tooltipenabled' property	282
The 'type' property	282
The 'unicode' property	283
The 'units' property	283
The 'update' property	284
The 'url' property	285
The 'usealtscale' property	285
The 'user' property	285
The 'valid' property	286
The 'value' property	286
The 'values' property	287
The 'verbs' property	288
The 'version' property	288
The 'view' property	289
The 'visible' property	289
The 'volume' property	290
The 'where' property	290
The 'winptr' property	291
The 'workarea' property	291
The 'wrap' property	292
The 'wssize' property	292
The 'xaltintercept' property	292
The 'xaxislabel' property	293
The 'xclasses' property	293
The 'xintercept' property	294
The 'xlabels' property	295
The 'xlogscale' property	295
The 'xmajorticks' property	296
The 'xminorticks' property	296
The 'xscale' property	296
The 'xvalues' property	297
The 'yaltaxislabel' property	297
The 'yaltlabels' property	297

The 'yaltlogscale' property	298
The 'yaltmajorticks' property	298
The 'yaltminorticks' property	298
The 'yaltscale' property	299
The 'yaxislabel' property	299
The 'yintercept' property	299
The 'ylabels' property	300
The 'ylogscale' property	300
The 'ymajorticks' property	301
The 'yminorticks' property	301
The 'yscale' property	301
The 'yvalues' property	302
Δ.. Delta property	302
Section 4. System Classes: Methods	305
The 'Abort' method	307
The 'Accept' method	307
The 'Addimages' method	308
The 'Addrows' method	309
The 'Arrange' method	309
The 'Back' method	309
The 'Beginlabeledit' method	309
The 'Cancellabeledit' method	310
The 'Chartalttopoint' method	310
The 'Chartoline' method	311
The 'Charttopoint' method	311
The 'Clear' method	312
The 'Click' method	312
The 'Clienttoscreen' method	313
The 'Close' method	313
The 'Closedocument' method	314
The 'Copy' method	315
The 'Create' method	315
The 'Cut' method	315
The 'Delete' method	316
The 'Deletecols' method	316
The 'Deletemessage' method	317
The 'Deletenodes' method	317
The 'Deleterows' method	318
The 'Doverb' method	318
The 'Draw' method	319
The 'Editstart' method	320
The 'Eject' method	320
The 'Ensurevisible' method	320
The 'Execute' method	321
The 'Expand' method	322
The 'Find' method	322
The 'Findnode' method	322
The 'Focus' method	323
The 'Forward' method	323
The 'Get' method	324
The 'Getinfo' method	325

The 'Getmessage' method	327
The 'Getsummary' method	327
The 'Head' method	328
The 'Hide' method	329
The 'Hittest' method	329
The 'Insertcols' method	330
The 'Insertnodes' method	330
The 'Insertrows' method	331
The 'Interrupt' method	331
The 'Job' method	331
The 'Linelength' method	332
The 'Linetochar' method	332
The 'Listen' method	332
The 'Load' method	333
The 'New' method	333
The 'Open' method	335
The 'Overlay' method	336
The 'Paint' method	340
The 'Paste' method	340
The 'Play' method	341
The 'Pointtocell' method	341
The 'Pointtochart' method	342
The 'Pointtochartalt' method	342
The 'Popup' method	343
The 'Post' method	343
The 'Poster' method	344
The 'Preview' method	344
The 'Print' method	345
The 'Receive' method	345
The 'Refresh' method	346
The 'Reset' method	346
The 'Resize' method	347
The 'Rewind' method	347
The 'Save' method	347
The 'Screentoclient' method	348
The 'Send' method	349
The 'Sendmessage' method	349
The 'Set' method	350
The 'Setimages' method	350
The 'Setinfo' method	350
The 'Setopacity' method	351
The 'Setpos' method	351
The 'Setup' method	352
The 'Show' method	352
The 'Showaboutbox' method	353
The 'Shownode' method	353
The 'Showproperties' method	354
The 'Signal' method	354
The 'Sortchildren' method	355
The 'Stepit' method	355
The 'Stop' method	356

The 'Transform' method	356
The 'Trigger' method	361
The 'Undo' method	361
The 'Valueof' method	362
The 'Wait' method	362
Section 5. System Classes: Callbacks	363
The 'onAboutMenu' callback	365
The 'onActivate' callback	365
The 'onChange' callback	365
The 'onClick' callback	366
The 'onClose' callback	366
The 'onColMoved' callback	367
The 'onConnectRequest' callback	367
The 'onDbClick' callback	367
The 'onDeactivate' callback	368
The 'onDestroy' callback	368
The 'onDisconnect' callback	368
The 'onDragDrop' callback	368
The 'onDragEnd' callback	369
The 'onDragEnter' callback	369
The 'onDragLeave' callback	370
The 'onDragOver' callback	370
The 'onDragStart' callback	371
The 'onDraw' callback	371
The 'onDropDown' callback	371
The 'onError' callback	371
The 'onExecute' callback	372
The 'onFocus' callback	372
The 'onHide' callback	373
The 'onKeyDown' callback	373
The 'onKeyPress' callback	373
The 'onKeyUp' callback	374
The 'onLimit' callback	374
The 'onMenu' callback	374
The 'onMouseDown' callback	375
The 'onMouseMove' callback	375
The 'onMouseUp' callback	376
The 'onMove' callback	376
The 'onMovieEnd' callback	376
The 'onOpen' callback	376
The 'onPaint' callback	377
The 'onPopup' callback	377
The 'onPreferencesMenu' callback	377
The 'onQuitMenu' callback	378
The 'onReady' callback	378
The 'onReceive' callback	379
The 'onResize' callback	379
The 'onRowMoved' callback	380
The 'onScroll' callback	380
The 'onSelChange' callback	380
The 'onSelection' callback	381

The 'onSend' callback	381
The 'onShow' callback	381
The 'onSignal' callback	381
The 'onStop' callback	382
The 'onTimer' callback	382
The 'onUnfocus' callback	382
The 'onZoom' callback	382
Section 6. Using the Draw Method	385
Using the Draw method	387
Draw method: State commands	389
Draw method: Rendering commands	394
Draw method: Grouping and Control commands	399
Section 7. Using the Chart and Series Objects	401
Chart and Series Objects: Introduction	403
Using the Chart Object	405
Axes, Coordinates and Scales	409
Chart Object Properties	413
Chart Object Methods	423
Series Object Properties	425
Customizing the Chart using the Draw method	430
Section 8. OLE, OCX and ActiveX Programming	432
OCX/ActiveX Controls and OLE Automation	433
Using OCX/ActiveX Controls	435
Using OLE linked/embedded documents	439
Using OLE (COM) Server Applications	444
Array Properties and Constants	446
Section 9. APLX Multi-tasking	447
Introduction to APLX multi-tasking	449
Creating APL child tasks under program control	452
Using the child task	454
Communication between child and parent tasks	457
Signal events	458
Using 'delta' properties to share data between tasks	460
Sharing variables between tasks	461

Section 1. Introduction to APLX System Classes

Introduction to System Classes

Overview

Built in to all versions of APLX (excluding console-only versions) is a rich set of *System Classes*, which implement windows, dialogs, graphics, movies, and other user-interface components which you can use from your APL applications. They also include more advanced components, such as a chart-drawing class for business and scientific graphs, an image-manipulation class, and some non-visual classes for networking, e-mail and web access. As far as possible, these classes work cross-platform, so that, with a few exceptions, the same APL user-interface code will run under APLX for Windows, MacOS or Linux.

The syntax for using these System Classes is similar to the syntax for user-defined classes (which you write yourself in APL), or external classes (for example, classes written in C# or Java). There is also an alternative syntax (based on the `□WI` system function) which is retained for compatibility with other APL interpreters and with earlier versions of APLX, and which is sometimes useful in itself.

The workspaces `10 HELPSYSCCLASS` and `10 SAMPLESYSCCLASS` contain examples of using System Classes. If you take a look at these examples, you will see that the basic steps which you typically need to carry out are as follows:

1. Create a top-level object (for example, a Dialog, Form, Window, or Document), using `□NEW`.
2. Create controls such as edit boxes, buttons, or rectangles on the top-level object, using the `New` or `Create` method which is implemented for all System Classes.
3. Provide APL functions known as *callbacks* which get run when certain events occur (such as when a Button is pressed).
4. Call the `□WE` system function which waits for events, and where appropriate executes your callback functions.

Each object you create is an instance of a pre-defined class of objects which are built into APLX (or accessible as an external OCX class). You tailor the appearance and behavior of an object by setting *properties* for it, and you find out about an object's current state by reading its properties. Properties include things like the size and position of an object, its title, its color and font, the current text displayed in it, and so on. In most cases, when you set a property, the object immediately reacts accordingly (for example, if you change its `size` property, it is immediately re-sized). Some properties (such as the `data` property, an arbitrary APL array associated with the object) are valid for all objects. Many properties (such as `size`) are valid for several classes of object, and a few (such as the `text` property) for certain specific classes only. Most properties can be both set and read by your program, but some are read-only.

You can also call built-in functions associated with objects from APL (these are known as *methods*). These carry out operations such as hiding or closing the object.

Creating instances of System Classes using `NEW`

The first stage in using System Classes is to create a top-level object (i.e. an instance of a class such as `Window`, `GetMail` or `Image`) using `NEW`. The right argument is the name of the class, and the left argument should be `' '` to indicate that this is a System Class (rather than a user-defined or external class).

For example, you might create a window with the standard Dialog border and appearance:

```
dlg←' ' NEW 'Dialog'
```

This has created a new object, and returned a reference to that object which has been placed in the variable `dlg`.

You can then set and read properties, or call methods, of this new object using *dot notation*, where you refer to the property or method using a compound name in the form `ObjectRef.PropertyName` or `ObjectRef.MethodName`. For example, to set the `caption` property of the window (which sets the title of the window):

```
dlg.caption←'My first window'
```

Reading the value of a property is similar:

```
dlg.caption
My first window
```

To call a method which takes no arguments, you simply refer to the object and method:

```
dlg.Show
```

To call a method which takes arguments:

```
dlg.Clienttoscreen 8 12
158 180
```

Not all System Classes can be created as top-level objects. For example, a `Button` can be created only as the child of a `Window` or similar container, as described below. The classes which can be top-level objects are either windows, pre-defined dialogs, or invisible classes. The full list is:

```
APL ChooseColor ChooseDir ChooseFont Dialog Document Form GetMail HTTPClient Image
ImageList Menu MsgBox OpenFile Printer SaveFile SendMail Socket System Timer
Window OLE (COM) Server
```

Creating Child controls

Once you have a top-level object (such a window or form), you can typically create *child controls* (such as buttons, list boxes, and so on) on it. Because these controls cannot exist independently of the parent object, you cannot use `NEW` to create them. Instead, you use the `New` or `Create` methods. These take a right argument which is the name of the System Class which you want to create. The name of the control is specified using dot notation.

for example, to create an object called `Lst1`, of class `List`, on the window which we have just defined, you would enter:

```
dlg.Lst1.New 'List'
```

When you create a control in this way, the system chooses defaults for properties such as size and position. You can change these using dot notation in the normal way. For example, to set the `size` property of the list you just created to be 5 standard rows high and 30 columns wide (the object will immediately be re-sized):

```
dlg.Lst1.size←5 30
```

Put a set of choices (contained in the variable `NAMES`) into the list box by setting its `list` property:

```
dlg.Lst1.list←NAMES
```

Read back the selection which the user has made by reading the `value` property of the list box:

```
dlg.Lst1.value
```

2

Rules for Object Names

As can be seen in the above examples, the dot-notation syntax is hierarchical in that the first part (up to the first dot) represents the reference to the top-level object, created using `NEW`. The next part represents the child object name (created using the `New` or `Create` method of the System Class subsystem), if there is one. Potentially there may be further levels of hierarchy separated by further dots - for example, a menu item may be a child of a sub-menu which is a child of a menu-bar item. The maximum length of each part of an object's name is 32 characters. Apart from the special significance of dots, names follow the normal APL symbol-naming rules. Object names are case-sensitive.

Finally, the last part of the dot-notation sequence is the property or method name. The valid property, method, and class names are pre-assigned and are documented separately. APLX ignores case when searching for these pre-assigned names, but it is recommended that you enter them as shown for future compatibility with other systems.

Responding to events

The processing of events which occur in your program (such as a button being pressed) is handled by the system function `WE` in conjunction with callback properties which you set. These callback properties tell the system 'If this event occurs for this object, then you should execute the following APL expression'. For example, if you want to run a function called `HITME` when a button is pressed, you would set the `onClick` handler for that button as follows:

```
MyWin.MyButton.onClick←'HITME'
```

The actual execution of the `HITME` function takes place during an event loop, invoked by the function `WE`. This is described in detail separately.

Drawing pictures, shapes and text

As well as placing controls and text or graphic objects on your windows, you can also use the `Draw` method to display text, geometric shapes such as polygons and ellipses, and pictures or bitmaps.

Implementing Drag-and-Drop

If you want to use drag-and-drop for specific functionality in your application, you can do this very simply. In outline, all you need to do is the following. First define the `sourceformats` property for the control which can be dragged. Then define the `targetformats` property for the control or controls on to which it can be dragged. If the two formats match, the user will now be able to drag the source control on to the target control. Your program will be notified when a drag-drop occurs using the `onDragDrop` callback for the target, and the `onDragEnd` callback for the source control.

Note: Under Windows, APLX Edit and RichEdit controls have built-in drag-and-drop capability for text editing, and you do not need to do anything for users of your application to take advantage of this.

Deleting objects

In order to free up memory used by an object, you must make sure it is deleted when you are finished with it. As with user-defined APL classes, a top-level object will be deleted when the last reference to it is deleted from the workspace. Thus, if the variable `dlg` in the above example was erased (either explicitly, or because the name was localized in a function header, and the function completed), then the window would be closed (if it were still open), and it would be deleted. All child objects on the window are automatically deleted when the parent is deleted.

You can also delete an object explicitly using the `Delete` method (note that this is not the same as closing the object). This closes the object (and any children it has), and frees up any memory which it was using. However, this does not erase any references to the object in your workspace, if it is a top-level object:

```
dlg.Delete
dlg
[[]:UNKNOWN OBJECT]
```

As a convenience, windows which have no `onClose` callback defined are automatically deleted if the user closes the window.

Alternative syntax using `OWI`

As an alternative to using `OWNEW` and dot-notation, you can interface to System Classes by means of the system function `OWI`. This syntax usually less readable than dot-notation, but is retained for compatibility with earlier versions of APLX and with some other APL interpreters. In a few cases, it is preferable to dot-notation.

When you use `OWI` to create objects, there is no object reference held in the workspace. Instead, you provide a name for the object, which is held in the System Class sub-system, and you use a character vector to identify the object.

The left argument of `⌈WI` is the name of the object, and the right argument names the property or method you wish to access, and if appropriate also provides the value you wish to assign to the property or pass to the method. When you create an object, you can optionally set properties at creation time, otherwise the system chooses defaults. You can use the `New` or `Create` methods to create a new object. The following examples show how this works:

Create a top-level object called `Example`, in this case a window with the standard `Dialog` border and appearance:

```
'Example' ⌈WI 'New' 'Dialog'
```

Create an object called `Lst1`, of class `List Box`, on the window `Example` (the system chooses defaults for things like size and position):

```
'Example.Lst1' ⌈WI 'New' 'List'
```

Set the `size` property of the list you just created to be 5 standard rows high and 30 columns wide (the object will immediately be re-sized):

```
'Example.Lst1' ⌈WI 'size' 5 30
```

Create an object called `Lst1`, as above, but this time set the `size` property at the time you create it:

```
'Example.Lst1' ⌈WI 'New' 'List' ('size' 5 30)
```

Put a set of choices (contained in the variable `NAMES`) into the list box by setting its `list` property:

```
'Example.Lst1' ⌈WI 'list' NAMES
```

Read back the selection which the user has made by reading the `value` property of the list box:

```
'Example.Lst1' ⌈WI 'value'
```

2

The exact syntax of `⌈WI` for the three possible cases is as follows:

Setting a property:

```
ObjectName ⌈WI PropertyName Value
```

`ObjectName` and `PropertyName` are character vectors, and `Value` can be any APL array valid for the particular property in question. The right argument to `⌈WI` is thus a two-element nested vector. However, as a convenience `⌈WI` accepts any length of nested array vector and treats the first element as the property name and the rest of the argument as the property value, so that the following two statements are both valid and do exactly the same thing:

```
'Win1.But' ⌈WI 'where' (12 20 3 8)      a Two-element vector
'Win1.But' ⌈WI 'where' 12 20 3 8      a Five-element vector
```

(You can also use the `New`, `Create` and `Set` methods to set multiple properties in one line.)

Reading a property:

```
Value ← ObjectName □WI PropertyName
```

ObjectName and PropertyName are character vectors as before, and the current value for the property is returned as the explicit result of □WI. For example:

```
'Win1.But' □WI 'where'
12 20 3 8
'Win1.But' □WI 'caption'
Cancel
```

Invoking a method:

```
{Result ← } ObjectName □WI MethodName {Argument}
```

ObjectName and MethodName are character vectors, and Argument is an optional argument to the method. It can be any APL array valid for the particular method in question. Again, □WI accepts any length of nested array vector and treats the first element as the method name and the rest as the argument to the method. In addition some methods return a result.

```
'Win1.Movie' □WI 'Rewind'           ⍝ Method with no argument
'Win1.But' □WI 'New' 'Button'       ⍝ Argument is 'Button'
'Win1.RichEdit' □WI 'LineLength' 4  ⍝ Argument is 4
17                                   ⍝ Method returns result
```

Mixing dot-notation and □WI

If you create a top-level object using □WI, you cannot access it using dot notation because there is no reference to the object held in the workspace. However, if you create a top-level object using □NEW, then you can access it (and its children, if any) using □WI. When an object is created using □NEW, APLX automatically assigns a name in the form `SYSOBJ-NNN` to the object, and you can use this name in the left argument to □WI. The easiest way of doing this is to reference the `self` property, which gives the fully-qualified name:

```
dlg←'□' □NEW 'Dialog'
dlg.self
SYSOBJ-1
dlg.Lst1.New 'List'
dlg.Lst1.self
SYSOBJ-1.Lst1
dlg.Lst1.self □WI 'List' □W
```

This is particularly useful for a few cases where □WI provides an easier syntax than dot-notation. These include accessing array properties in OCX/ActiveX controls, and dynamically determining the source of an event using □WSELF.

Converting □WI syntax to dot-notation

If you have written code using □WI syntax in previous versions of APLX, it will continue to work. However, if you wish, you can convert the syntax to use □NEW and dot-notation. The workspace 10 `CONVERTQWI` helps to automate this.

Special considerations for Client-Server versions of APLX

In Client-Server implementations of APLX, the front-end which implements the user-interface (the "Client") runs on one machine, and the APLX interpreter itself (the "Server") can run on a different machine. The two parts of the application communicate via a TCP/IP network. Typically, the Client will be the APLX front-end built as a 32-bit Windows application running on a desktop PC, and the Server will be a 64-bit APLX64 interpreter running on a 64-bit Linux or Windows server. Alternatively, the Client and the Server may run on the same physical machine (this is the case if you are running APLX64 on a 64-bit desktop machine).

In this environment, it is important to bear in mind that **the whole of the System Class sub-system runs on the Client machine as a 32-bit application**. This is nearly always transparent to your application, so for example a workspace originally written for a 32-bit Windows desktop machine should continue to work unchanged in a Client-Server configuration where the Client executes on a 32-bit Windows desktop and the APLX interpreter (Server) executes on a 64-bit server machine. However, you should be aware that any file or path names used in your objects always relate to the Client machine, never to the Server.

In such an environment, when you make a `⌵WI` call, the parameters are passed over the network from the Server to the `⌵WI` sub-system on the Client machine. If the Server is a 64-bit APLX64 interpreter, the data will be converted to 32-bit form before it is sent over the network. (This means that any data you associate with an object using the `data` property or a 'delta' variable will be held on the Client machine, and must be representable as a 32-bit variable).

See also the description of the `APL (Child Task)` object for special considerations applying to multiple tasks in a Client-Server environment.

System Classes by category

Windows and Menus

To create a top-level window on which other controls will be placed, you normally use the Window or Form class. The Dialog class is similar, but is automatically preset with the default appearance and behavior suitable for a dialog box. You can add menus to your windows using the Menu class.

The Document class is a convenience class which comprises a window with a built-in text-edit capability.

Buttons, Combos, and List Boxes

Simple controls which you will use on many dialogs include the Button class for standard action buttons, the Check and Radio buttons which allow users to select options, and the Combo box. The List class implements standard List boxes, for selecting one or more items from a list.

The ToolButton class can be used as an alternative to Buttons, Check boxes and Radio buttons. It allows you to specify an image to be displayed on the button, and is particularly suitable for implementing a toolbar.

Displaying and inputting text

You can use the Label class to place static text on your windows. For editable text, use the Edit class; this can handle both single-line and multi-line text input and editing.

For more sophisticated text display and editing, with full control over formatting, you can use the powerful RichEdit class.

(See also the Draw method, which provides an alternative way of placing text on windows).

Selecting or displaying a numeric value or position

The Spinner class allows the user to select a numeric value using a pair of up-down or left-right arrows. The Trackbar class displays a slider, which the user can move to select a value. The Scroll class implements a scroll bar.

The Progress class implements a progress bar; it is typically used to show the progress of a long operation.

Trees and Grids

Two of APLX's built-in classes provide more complex user-interface functionality, although they are still easy to use. These are the Tree object class, for displaying and manipulating hierarchical tree-

views of data, and the Grid class, which provides a spreadsheet-like interface for handling arrays of data.

Shapes and Icons

These controls provide simple decorative elements for your dialogs. They include the Line, Rectangle, RoundRect, Arc and Icon classes. (You can also use the Draw method for more sophisticated geometric drawing.)

Pictures, Images and Movies

You can display pictures, such as JPEG and BMP files, using a Picture object. This can also be used to display APL arrays of pixel values.

For more sophisticated image handling, APLX provides an interface to the ImageMagick package, encapsulated as the Image class. This provides the capability for loading and saving graphics images in a very wide range of formats, as well as providing facilities for scaling, transformation, color adjustment, and image enhancement. It can also be used in conjunction with the Draw method for creating or adding graphics elements to images.

Movies can be played using the Movie class.

The ImageList class is a special non-visual class which holds images displayed in menus and other controls. It is not available under MacOS.

Charts

The Chart object class, together with the related Series class, makes it very easy to add business and scientific graphs to your APLX applications. It can display Line, Area, Scatter, Stair, High-Low-Open-Close, Candlestick, Bar, Stacked Bar, and Pie charts.

Geometry management and tabs

The Bevel and Frame classes allow you to group controls together visually. For complex dialogs, you can use a Splitter control to allow the user to expand or contract regions of the dialog. You can also use the Selector and Page classes to implement tabbed dialogs.

Pre-defined dialogs

APLX includes a number of pre-defined dialogs, which encapsulate all of the necessary functionality for common tasks such as displaying message boxes (MsgBox), selecting files and directories (OpenFile, SaveFile and ChooseDir), selecting colors (ChooseColor), and (under Windows and Linux only), selecting fonts (ChooseFont). These are top-level objects, which should not be created as a child of a window, but can be used directly.

Non-visual classes

A few special classes do not represent visible controls or windows. These include the System class (which is created automatically as the parent of all top-level objects, and which can be used to return information about the system APLX is running on and for accessing the Clipboard), the Timer class (which runs APL callbacks at specified intervals), and the Printer class (for printing text and graphics).

Networking classes

Another set of non-visual classes allows your APLX applications to exchange data over a local-area network or the Internet. These include classes for sending and retrieving e-mails (SendMail and GetMail), for retrieving web pages and other documents using the HTTP protocol (HTTPClient), and the low-level Socket class.

OCX/ActiveX (COM) classes

Under Windows, as well as using the APLX built-in classes such as Button and MsgBox, you can also use external OCX/ActiveX controls (also known as COM controls). You can also embed or link to documents belonging to other applications in an OLEContainer control, or invoke a separate application such as Microsoft Excel and exchange data with it. See the section on OCX/ActiveX Controls and OLE Automation for more information.

Multi-tasking

Using the APL class, your applications can create and communicate with new APL child tasks. These can either be background tasks, or have their own Session windows.

Positioning controls in windows

Setting the initial position of a control

When you create a control on a window, you normally set its position using the `where` property. This defines the initial position of the control relative to the top left of the window, in units set by the `scale` property of the parent. This example creates a window with a multi-line text-edit box taking up most of the window, and an OK button in the right-hand corner:

```
Win1←'□' □NEW 'Window' ♦ Win1.where←4 4 14 28
Win1.Ed.New 'Edit' ♦ Win1.Ed.where←1 1 8 25 ♦ Win1.Ed.style←8196
Win1.But.New 'Button' ♦ Win1.But.where←10 16 ♦ Win1.But.caption←'OK'
```

Caution: For compatibility with other APL systems, the `scale` property defaults to 1, 'character' units, which is not necessarily the most convenient scale to use. You can change this to 5 (pixels) for more precise positioning, or to values expressed in physical units such as points or millimetres. The `scale` property for a given object is inherited from its parent when you create it, so it is often best to set the `scale` of the System object (which is the ultimate parent of all objects) at the beginning of your program. See the description of the `scale` property for more details.

If you want to query or change the size of a control but not its position, you can use the `size` or `extent` properties. These are the same, except that `size` is expressed in the object's own `scale`, whereas `extent` is in the `scale` of its parent.

Handling re-sizing

If the window or dialog can be re-sized by the user, you need to consider what should happen to the control positions and sizes as the window size changes. The default is for the controls to remain at their initial positions, and to stay the same size. For some controls, that may be what you want. In many cases, however, you will want controls such as text-edit fields to get larger or smaller as the window changes size. You may also want controls such as buttons to move so that they remain anchored to the right or bottom-right of the window. In our example above, we want the OK button to remain at the bottom-right of the window, and we want the edit field to grow or shrink with the window.

One way of achieving this is to write an `onResize` callback, i.e. an APL function which gets called when the window is resized. Your APL function can then read the new window size, and move the controls around as necessary.

However, a much easier way is to use the `anchors` property. This defines which edges of the window the control is anchored to. It comprises a vector of four boolean values, which govern whether the control is anchored to the top, left, bottom and right of the window respectively. The default is `1 1 0 0`, meaning that the control is anchored to the top and left of the window. As the window is resized, the position of the control remains fixed relative to the top left.

If you set the `anchors` property to `0 0 1 1`, the control will be anchored to the bottom and right of the window. This means that, as the window is resized, the control will remain at a constant distance from the bottom right corner of the window (its `where` property will change accordingly). This is exactly what we want for our OK button in the above example:

```
Win1.But.anchors←0 0 1 1
```

If a control is anchored to both the left and the right, then as the window is re-sized, the left edge of the control will remain at a fixed distance (as initially set by the `where` property) from the left edge of the window, and the right edge of the control will remain at a fixed distance from the right edge of the window. Thus the control will grow or shrink horizontally with the window. Similarly, if the control is anchored to both the top and the bottom, it will grow or shrink vertically with the window. This is what we want for our text-edit field in the above example:

```
Win1.Ed.anchors←1 1 1 1
```

Forcing a control to align with an edge of its parent

Another way of managing control positions and sizes is to use the `align` property. This is somewhat like the `anchors` property, in that it governs the position of a control by reference to the edge of the parent. However, unlike the `anchors` property, it overrides the `where` property, and forces the control to be positioned hard against the edge. It also forces it to expand or contract to be the same height or width as the parent. You can set it to one of five values:

```
0 No alignment (default)
1 The control is aligned along the top edge of its parent
2 The control is aligned along the left edge of its parent
3 The control is aligned along the bottom edge of its parent
4 The control is aligned along the right edge of its parent
~1 The control fills the whole client area of its parent
```

If, instead of the above example, you wanted the text edit control to fill the whole window, you could write:

```
Win1←'□' □NEW 'Window' ♦ Win1.where←4 4 14 28
Win1.Ed.New 'Edit' ♦ Win1.Ed.align←~1 ♦ Win1.Ed.style←8196
```

Using Splitter controls

A special case arises with the Splitter control. This is a vertical or horizontal line control, which allows the user to change the relative sizes of the controls on a window. For example, you might have a list box and a text edit, and use a Splitter to allow the user to move the boundary between them.

To do this, you first set up the leftmost (or topmost) control or controls, using the `align` property to align to the left (or top). You then create the Splitter, defining its position with the `where` property. Then you create the rightmost (or bottom) control, setting its `align` property to fill the remaining client area of the parent:

```
DEMO←'□' □NEW 'Window'
DEMO.title←'Splitter Example' ♦ DEMO.scale←5
DEMO.List1.New 'List'
```

```
DEMO.List1.align←2 ♦ DEMO.List1.size←50 20 ♦ DEMO.List1.list←DM
DEMO.MySplitter.New 'Splitter' ♦ DEMO.MySplitter.where←0 22
DEMO.Edit1.New 'Edit' ♦ DEMO.Edit1.style←36 ♦ DEMO.Edit1.align←1
```

Setting constraints on the maximum and minimum window size

Often, you may want to put limits on the maximum or minimum height and width of a window, so that the user cannot make it so small that it becomes unusable, or so large that it wastes screen space. The `maxsize` and `minsize` properties allow you to specify the maximum and minimum sizes respectively.

Automatic adjustment of control sizes for MacOS Aqua

The new look-and-feel (known as 'Aqua') which Apple introduced in MacOS X can sometimes require controls to be slightly larger than they were in the 'classic' MacOS environment. For example, Buttons have a more rounded look, which for some dialogs can mean that text which used to fit in the button is now partially obscured. If you have code designed for the 'classic' MacOS which you now want to run under MacOS X, the sizes therefore may need adjustment. Obviously, you can go through your code and adjust the sizes of the controls on each dialog, but this can be very time-consuming. APLX provides a quick method of reducing the conversion effort. If you set the `aquaadjust` property for the System object, all subsequent Buttons, Radio buttons and Checkboxes which are created will be adjusted in size slightly (although some additional manual adjustment may still be needed). This property is inherited by window objects, as well as the individual controls, so you can set it on or off individually for particular dialogs or controls. The property is ignored under Windows and Linux.

Events and Callbacks

Concepts

Windowing systems such as MacOS and Windows are event-driven, which means that your program has to be able to respond to events such as a key stroke or new data being available. An event (sometimes called a message) can arise from a number of sources. It can be directly triggered by a user action - for example, Mouse Down or Key Down. It might be indirectly triggered by a user action - for example, if the user presses the mouse button over a button in a dialog, a Mouse Down event occurs, and this in turn causes a Clicked event for the button. Events can also be triggered by the system software - for example, to indicate that a window needs to be redrawn - or by another program or even your own program.

Much event handling is automatic, and you do not normally need to deal with it. For example, redrawing of windows is usually handled automatically by APL. For events which you do need to know about, such as a Click event in a button, you define an APL *callback*, which is an APL expression (usually a function) which will be run if the event occurs. You assign a character vector containing the expression or function name to the appropriate callback property of the object. This is a special property (whose name begins with 'on..') which creates the association between the event and your callback function or expression.

For example, this code creates a dialog with an OK button, and associates the APL function ENDDIALOG with the Click event for the button:

```
DEMO←'□' □NEW 'Dialog'
DEMO.title←'Button Example'
DEMO.OK.New 'Button'
DEMO.OK.onClick←'ENDDIALOG'
```

The APL callback function is actually run during execution of the system function □WE (wait for event).

How APLX handles events

When an event occurs, APLX looks to see if the event refers to a user-defined window, or to one of APL's own windows (such as a session or edit window). In the latter case, the event is handled internally and you do not normally need to be aware of it. If the window was user-defined, however, then your APL application might need to be informed of the event. For every event, there is a default system action (which may be to do nothing), and there might in addition be an APL callback function which you have defined for that event. Often, the default system action is to create another, higher-level event. At the lowest level, events take place in windows, but where appropriate are passed to a control (such as an Edit text or Check box) to produce a higher-level event.

When events are handled

APLX calls the operating system for the latest event at two main points in the code. These are:

- (a) When APL is requesting input
- (b) During execution, the APL interpreter many times a second to see if any events need to be handled.

The system handler for an event is invoked immediately the event is retrieved from the operating system. This means that events continue to be processed even if your application is busy (doing a long calculation, for example), which helps APLX applications appear responsive to window updates and user actions. In contrast, your APL callback functions are executed only when you call `⎕WE`.

Where you have defined an APL callback for an event, the event is placed in a queue, and callbacks in the queue are run in turn when your program next calls `⎕WE`. You can either call `⎕WE` in an 'event loop' (with a long or infinite timeout value), in which case all processing takes place by `⎕WE` looping internally and executing your callback functions as events happen, or you can call `⎕WE` with a small or zero timeout value, in which case it will return almost immediately and you can continue with other APL processing. Where your program needs to carry out processing which will take a long time, it should call `⎕WE` from time to time (preferably at least two or three times a second), bearing in mind that when it does so events may cause other callbacks to be invoked.

Note that, when APL is in desk-calculator mode (for example, if an untrapped APL error occurs in your program), your APL callback functions will not be run. When you next call `⎕WE`, there may be quite a few callbacks queued up. You can flush these from the event queue by passing a left argument of 1 to `⎕WE`.

Information about events

When your APL callback function runs, you can find out information about the event using the niladic system function `⎕EV`. This returns the system clock, keyboard state and mouse position at the time of the event, a number which identifies the event type, and sometimes extra information associated with a particular type of event.

Modal Dialogs

Making your own modal dialogs

The default behaviour of all windows you create (even Dialog objects) is non-modal, which means that the user can activate other windows and menus. Often, this behaviour is preferred, since it gives the user more control over the application. Sometimes, however, you may wish a dialog to be modal, which means that the user must close the dialog (typically by clicking in an OK or Cancel button) before continuing with other tasks. In addition, you might want your program to wait until the dialog has been closed before proceeding. These two features are logically separate, but often go together. You make the dialog modal by invoking the `Wait` method on it, and you cause your program to wait for the dialog to be closed by calling `⌈WE` with a right argument which is the name of the window. The following is a complete example of a function which puts up a modal dialog containing a label and two buttons. The function then waits until the user has responded, and returns 1 if the user presses the 'Yes' button or 0 if he presses the 'No' button.

```

    ▽R←ASK;X;MyDlog
[1] ⍝ Example of a modal dialog
[2] MyDlog←'⌈' ⌈NEW 'Dialog' ⋄ MyDlog.size←5 34
[3] MyDlog.Label.New 'Label' ⋄ MyDlog.Yes.where←0.5 4 1.5 30
[4] MyDlog.Label.caption←'Do you want to erase this file?'
[5] MyDlog.No.New 'Button' ⋄ MyDlog.Yes.where←3 3 1.5 8 ⋄ MyDlog.Yes.style←2
[6] MyDlog.Yes.New 'Button' ⋄ MyDlog.Yes.where←3 23 1.5 8 ⋄ MyDlog.Yes.style←1
[7] MyDlog.No.onClick←'R←0 ⋄ MyDlog.Close'
[8] MyDlog.Yes.onClick←'R←1 ⋄ MyDlog.Close'
[9] MyDlog.Wait
[10] R←0
[11] X←⌈WE MyDlog
    ▽

```

In this function, line [9] causes the dialog to be modal, and line [11] causes the function to wait until the Dialog object `MyDlog` has been closed (or hidden), executing any callbacks which are invoked as a result of events that come in. The Dialog object can be closed under program control, by calling the `Close` method. In this example, this happens when either of the Yes or No buttons is clicked, which causes the `onClick` handler to be invoked, as set up in lines [7] and [8]. If the user presses the 'Yes' button, the APL function behaves very much as though line [11] was:

```
[11] ⍝ 'R←1 ⋄ MyDlog.Close'
```

which has the effect both of setting the explicit result of the function and closing the dialog.

Pre-defined modal dialogs

As an alternative to creating your own modal dialogs, you can often use one of the pre-defined dialog objects which are built-in to APLX. These include `MsgBox`, `ChooseFont`, `ChooseDir`, `ChooseColor`, `OpenFile` and `SaveFile`. To use these, you create an instance of the object, set the properties you want, and then call the `Show` method. This displays the dialog modally, and returns an integer indicating which button was used to exit. Where appropriate, you can then read the properties to see what the user selected. The above example can be written more simply as:

```
    ▽R←ASK;MBOX
[1]  ⚡ Modal dialog using the MsgBox object
[2]  MBOX←'□' □NEW 'MsgBox' ♦ MBOX.style←3 ♦ MBOX.icon←1
[3]  MBOX.text←'Do you want to erase this file?'
[4]  R←6=MBOX.Show
    ▽
```


Section 2. List of APLX System Classes

This section lists the System Classes implemented by APLX, in alphabetical order. Subsequent sections list the properties, methods and events which APLX supports.

APL (Child task) Object

Description

The `APL` object allows you to create APL child tasks under program control. They can either be *background* tasks (with no session window), or alternatively they can be ordinary tasks with their own session windows, although these might be hidden. Child tasks may be terminated in the same ways as top-level tasks, but in addition they can be terminated under program control by the parent task. They will also terminate automatically if their parent task terminates. Child tasks can themselves create further child tasks.

Each task has its own separate memory allocation for the workspace, of a size which you can specify. The tasks execute independently, but you can use *signals* to allow parent and child tasks to communicate with one another. In addition, variables can be shared between tasks. The niladic system function `⊞UL` returns the number of APLX tasks which are currently running.

If you are running a Client-Server implementation of APLX, you can set the `host` property of the `APL` object to specify on which machine the new APL task should run.

See the separate section on APLX Multi-tasking for more details.

Example

```

▽DEMO_APL;ChildTask
[1]  ⍎ Sample function demonstrating use of the APL object
[2]  ChildTask←'⊞' ⊞NEW 'APL'
[3]  ChildTask.wssize←100000
[4]  ChildTask.Open
[5]  ChildTask.Execute '"There are now "',(⊞⊞UL)," APLX tasks running"'
[6]  0 0ρ⊞DL 5
[7]  ⍎ ChildTask will be deleted when function ends
▽

```

Properties

background children class data events extent host methods name opened port properties scale self size status taskid text tie visible where wssize

Methods

Close Create Delete Execute Hide Interrupt New Open Send Set Show Signal Trigger

Callbacks

onClose onDestroy onError onExecute onOpen onReady onSend onSignal

Arc

Description

The Arc class implements the circles, ellipses and arcs. They can be filled or just outlined.

These objects are normally used for display purposes only. The `where` or `size` property is used to define the enclosing rectangle.

The `angle` property defines the start and end angle; if this is 0 0, a complete circle (or ellipse) is drawn.

You can set the `filled` property to indicate that you want the object filled with the foreground color. You can set the foreground color using the `color` property. (The background color is ignored). The `pensize` property changes the thickness of the lines.

Note: Remember that the default scale is in character units. See the `scale` property for details.

See also the `Draw` method which allows you to draw geometric shapes on your windows and controls.

Example

```

▽DEMO_Arc;DEMO
[1]  ⍺ Sample function demonstrating use of the Arc object
[2]  DEMO←'⎕' ⎕NEW 'Window' ⋄ DEMO.scale←1
[3]  DEMO.title←'Arc Example'
[4]  ⍺
[5]  DEMO.ARC.New 'Arc'
[6]  DEMO.ARC.where←4 4 3 5
[7]  DEMO.ARC.color←255
[8]  DEMO.ARC.angle←90 330
[9]  ⍺
[10] DEMO.FILLEDARC.New 'Arc'
[11] DEMO.FILLEDARC.where←4 12 3 5
[12] DEMO.FILLEDARC.color←(255×256)
[13] DEMO.FILLEDARC.angle←90 330
[14] DEMO.FILLEDARC.filled←1
[15] ⍺
[16] ⍺ Wait for the user to close the window
[17] 0 0⍑WE DEMO
▽

```

Properties

align anchors angle aquaadjust autodraw children class color data dragsource droptarget enabled events extent filled maxsize methods minsize name opened pen pointer properties scale self size sourceformats targetformats tie units visible where

Methods

Click Clienttoscreen Close Create Delete Hide New Open Paint Resize Screentoclient Send Set Show
Trigger

Callbacks

onClick onClose onDbIclick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave
onDragOver onDragStart onHide onMouseDown onMouseMove onMouseUp onOpen onSend
onShow

Bevel

Description

The Bevel class implements bevelled boxes and lines, typically used as separators and layout panels in dialogs. Use the `where` property to define the rectangle containing the bevel, and `style` property to set the type and appearance:

- 0 The entire client area appears lowered or raised
- 1 The client area is outlined by a lowered or raised frame.
- 2 The bevel displays a line at the top of the client area.
- 3 The bevel displays a line at the bottom of the client area.
- 4 The bevel displays a line at the left side of the client area.
- 5 The bevel displays a line at the right side of the client area.
- 6 The bevel is an empty space.

You can also optionally add 8 to this value, to make the bevel raised rather than lowered.

Note: A Bevel is a purely geometric object. It cannot be the parent of other controls.

Example

```

▽DEMO_Bevel;DEMO
[1]  ⌘ Sample function demonstrating use of the Bevel object
[2]  DEMO←'␣' ␣NEW 'Dialog' ⋄ DEMO.where←4 4 ⋄ DEMO.scale←1 ⋄ DEMO.size←16 40
[3]  DEMO.title←'Bevel Example'
[4]  ⌘
[5]  ⌘ Bevel as horizontal line
[6]  DEMO.Bevel1.New 'Bevel' ⋄ DEMO.Bevel1.where←2 0 1 40 ⋄ DEMO.Bevel1.style←2
[7]  ⌘
[8]  ⌘ Bevel as lowered rectangle
[9]  DEMO.Bevel2.New 'Bevel' ⋄ DEMO.Bevel2.where←4 1 8 38 ⋄ DEMO.Bevel2.style←0
[10] DEMO.Button.New 'Button' ⋄ DEMO.Button.where←6 15 3 12
[11] DEMO.Button.caption←'Action'
[12] ⌘
[13] ⌘ Wait for the user to close the window
[14] 0 0␣WE DEMO
▽

```

Properties

align anchors aquaadjust autodraw caption children class color data dragsource droptarget enabled events extent maxsize methods minsize name opened pointer properties scale self size sourceformats style targetformats tie units visible where

Methods

Click Clienttoscreen Close Create Delete Hide New Open Paint Resize Screenshotclient Send Set Show Trigger

Callbacks

onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave
onDragOver onDragStart onHide onMouseDown onMouseMove onMouseUp onOpen onSend
onShow

Browser

Exact behavior depends on operating system - see below

Description

The APLX Browser control implements a web browser, as a control which you can place on windows which you create. You can use it to display HTML Help files in your application, or to display pages from web sites accessed over the internet (*Note: Under Linux, this is only partially implemented, as described below*).

To use the Browser control, you create a window in the normal way, and place the Browser control on it. (If you want the Browser to fill the whole window, use the `align` property with a value of `^1`). You tell the control to fetch an HTML page (either from the local disk, or from the internet) by using the `Load` method. The user can then click on hyperlinks to load further pages, as in a normal web browser.

Browser properties

Key properties of the Browser control include:

`status`: (*Read-only*) Returns a three-element boolean vector. The first is 1 if the control is busy fetching a page, 0 if it is not busy. The second is 1 if the `Back` method can be called (i.e. there is a previous page in the history list), else 0. The third is 1 if the `Forward` method can be called (i.e. there is a subsequent page in the history list).

`text`: (*Read-only*) Returns a character vector (with embedded carriage returns), containing the current text as displayed in the control, with the HTML tags stripped out.

`contents`: (*Read-only*) Returns a character vector (with embedded carriage returns), containing the raw HTML text of the page currently displayed in the control.

`caption or title`: (*Read-only*) Returns a character vector containing the title of the page currently displayed in the control.

`url`: (*Read-only*) Returns a character vector containing the URL (Uniform Resource Location) of the page currently displayed in the control. This is not necessarily the same as the page which has been requested using the `Load` method or clicked-on by the user, because the request may have been redirected to a different location.

`offline`: If the `offline` property is set to 1, the Browser object will access only local files and cached copies of pages. The default is 0, meaning that the Browser will connect via the Internet to access web pages where necessary. *Note: This property is ignored under MacOS.*

Browser methods

Key methods of the Browser control include:

Load: The Load method takes a single parameter, which is the URL of the web page you want to display. It can be an internet address (such as 'http://www.bbc.co.uk/news'), or a local file address (such as 'file://C:\Program Files\MicroAPL\APLX\help\HTML\ch.htm'). You can also just specify a local file name without the 'file://' prefix. It is important to note that the actual loading of the page happens asynchronously, i.e. the Load method may return before the page has been retrieved from the internet. You can use the `status` property to determine whether the page is still being loaded.

Back: This method takes no arguments. It causes the Browser to go back one page in the history of previously-displayed web pages. It does nothing if there is no previous page. (You can test for this by reading the `status` property).

Forward: This method takes no arguments. It causes the Browser to go forward one page in the history of previously-displayed web pages. It does nothing if there is no subsequent page. (You can test for this by reading the `status` property).

Copy: This method takes no arguments. It causes the Browser to copy any selected text to the Clipboard.

Print: This method takes no arguments. It causes the Browser to print the currently-displayed page on the currently-selected printer. *Note: The Print method is not implemented under Linux.*

Stop: This method takes no arguments. If a page is loading, this method will abort the fetch.

Refresh: This method takes no arguments. It causes the current page to be reloaded from the internet or from disk, as appropriate.

Browser events

The Browser control can trigger the following callbacks:

onReady: This event occurs when a page has been successfully loaded and displayed.

onError: This event occurs when an error occurs retrieving a page.

Platform-dependent notes

Under **Windows**, the APLX Browser control is a wrapper for the Internet Explorer ActiveX Control. This means that, to use the Browser control, you must have Microsoft Internet Explorer installed. Note that, as well as using the properties and methods documented for the APLX Browser control itself, you can also directly access the Internet Explorer properties and methods (with names preceded by 'x' or 'X' respectively). See Microsoft's documentation on Reusing the Web Browser Control for more details on programming the browser directly.

Under **Linux**, the APLX Browser control is built-in to APLX, but its functionality is very restricted. It uses a limited HTML rendering engine which can handle simple HTML pages, but which does not have any scripting, plug-in, form-filling or encryption capability. Unrecognised HTML elements are ignored. In addition, the control implements only a small subset of the HTTP protocol. This means that many web pages will not display or operate correctly. The `Print` method is not supported. However, the Browser control is still very useful for adding HTML help to an application, and for certain accessing web pages which you know can be displayed reasonably well using only the simpler HTML elements. (The same underlying control is used to display the APLX Help pages under Linux).

Under **MacOS X**, the APLX Browser control is a wrapper for Apple's Safari HTML display engine. This is available under MacOS 10.2 and later only. Because of limitations in the Safari control, it may not operate correctly if the window contains Edit fields and other input controls.

Under **MacOS 9**, the APLX Browser control is not available.

Example

```

▽DEMO_Browser;DEMO
[1]  ⍺ Sample function demonstrating use of the Browser object
[2]  DEMO←'□' □NEW 'Window' ◇ DEMO.scale←5 ◇ DEMO.size←470 800
[3]  DEMO.Web.New 'Browser' ◇ DEMO.Web.align←¯1
[4]  ⍺
[5]  ⍺ Add a callback so window title changes when a new page is loaded
[6]  DEMO.Web.onReady←"DEMO.title←DEMO.Web.title"
[7]  ⍺
[8]  ⍺ Display a page from the MicroAPL web site
[9]  DEMO.Web.Load 'http://www.microapl.co.uk/apl'
[10] ⍺
[11] ⍺ Process events until the window is closed
[12] 0 0ρ□WE DEMO
▽

```

Properties

align anchors aquaadjust autodraw caption children class color contents data doublebuffered
 dragsource droptarget enabled events extent handle maxsize methods minsize name offline opened
 order pointer properties scale self size sourceformats status tabstop targetformats text tie units url
 verbs visible where winptr

Methods

Back Click Clienttoscreen Close Copy Create Delete Draw Focus Forward Hide Load New Open
 Paint Print Refresh Resize Screentoclient Send Set Show Stop Trigger

Callbacks

onClick onClose onDblClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave
 onDragOver onDragStart onError onFocus onHide onKeyDown onKeyPress onKeyUp
 onMouseDown onMouseMove onMouseUp onOpen onReady onSend onShow onUnFocus

Button

Description

The Button class implements the command button objects, like the OK and Cancel buttons found in many dialogs. You typically set the `caption` property which indicates the caption associated with the control, and you might set the `enabled` property to 1 or 0 depending on whether a particular option is valid (for example, an OK button might be disabled unless the user has entered some valid text in an Edit field)

For Buttons, you will always want to define an `onClick` callback, which is invoked when the button is pressed, and which in many cases will close the dialog. For most dialogs, you should define one of your buttons (typically the OK button or equivalent) to have a `style` property of 1, which means that it is the default button which ends the dialog. Buttons with a `style` of 1 are outlined in a heavy line, and respond to the Enter or Return key as well as to mouse clicks. In addition, you may wish to have a second button with a `style` of 2, which means that it is a Cancel style button which responds to the Escape or (under MacOS) Command-period key.

To provide help text when the user moves the mouse pointer over the control and pauses, you can use the `tooltip` property.

See also the ToolButton class, which provides a different kind of button on which you can place an image as well as or instead of text.

Example

```

▽DEMO_Button;DEMO
[1]  A Sample function demonstrating use of the Button object
[2]  DEMO←'□' □NEW 'Dialog' ◇ DEMO.where←5 10 ◇ DEMO.scale←1 ◇ DEMO.size←10 46
[3]  DEMO.title←'Button Example'
[4]  A
[5]  A CREATE OK BUTTON
[6]  DEMO.OK.New 'Button'
[7]  DEMO.OK.style←1
[8]  DEMO.OK.where←2 5 2 12
[9]  DEMO.OK.caption←'I'm OK'
[10] DEMO.OK.tooltip←'Press this button if you are feeling fine today'
[11] DEMO.OK.onClick←'□←''OK Hit!''
[12] A
[13] A CREATE CANCEL BUTTON
[14] DEMO.CANCEL.New 'Button'
[15] DEMO.CANCEL.style←2
[16] DEMO.CANCEL.where←2 20 2 20
[17] DEMO.CANCEL.caption←'You're Cancelled'
[18] DEMO.CANCEL.tooltip←'Want to cancel something? Now is your chance'
[19] DEMO.CANCEL.onClick←'□←''Cancel Hit!''
[20] A
[21] A Wait for the user to close the window
[22] 0 0□WE DEMO
▽

```

Properties

align anchors aquaadjust autodraw caption children class color data doublebuffered dragsource droptarget enabled events extent font handle maxsize methods minsize name opened order pointer properties scale self size sourceformats style tabstop targetformats tie tooltip units visible where winptr

Methods

Click Clienttoscreen Close Create Delete Draw Focus Hide New Open Paint Resize Screentoclient Send Set Show Trigger

Callbacks

onClick onClose onDbIclick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave onDragOver onDragStart onFocus onHide onKeyDown onKeyPress onKeyUp onMouseDown onMouseMove onMouseUp onOpen onSend onShow onUnFocus

Chart

Description

The Chart object is a control which can display a range of different business, statistical and scientific graphs. The actual data which is displayed in the graph is held in one or more Series objects.

The types of chart which can be displayed include Line, Area, Scatter, Bar, Stacked Bar, Horizontal Bar, Horizontal Stacked Bar, High-Low-Open-Close, and Candlestick graphs. You can also mix different graph types on a single Chart.

For full details, see the separate section on the Chart and Series Objects

Example

```

vDEMO_Chart;labels;W
[1]  a Sample bar chart (See workspace 10 SAMPLESCHART for more examples)
[2]  W←'□' □NEW 'Window' ♦ W.title←'Beer Consumption' ♦ W.visible←3
[3]  W.Ch.New 'Chart' ♦ W.Ch.align←1 ♦ W.Ch.type←'bar'
[4]  W.Ch.title←'Thirsty people'
[5]  W.Ch.subtitle←'Beer sales per head of population'
[6]  W.Ch.note←'Source: The Economist Pocket World in Figures, 2004'
[7]  W.Ch.yaxislabel←'Retail Sales per head, litres'
[8]  W.Ch.s1.New 'Series' ♦ W.Ch.s1.color←374479
[9]  W.Ch.s1.values←85.8 76.7 73.9 69.9 69.7 66.6 64.7 64.5 49 40.2 38.6 31.7
[10] labels←'Czech Republic' 'Venezuela' 'Germany' 'Denmark' 'S.Africa'
[11] labels←labels,'Austria' 'US' 'Australia' 'Canada' 'Belgium' 'Japan' 'UK'
[12] W.Ch.xlabels←labels
[13]  a
[14]  a Wait for the user to close the window
[15]  0 0ρ□WE W
v

```

Properties

align anchors aquaadjust autodraw axiswidth barwidth bitmap border caption children class color coloraxis colorgrid colorlegend colornote colortitle data doublebuffered dragsource droptarget enabled events extent font fontaxis fontlegend fontnote fonttitle gridwidth handle linewidth margin maxsize methods minsize monochrome name note opened order picture placelegend placenote placetitle pointer properties scale self size sourceformats style subtitle svg tabstop targetformats tie type units update visible where winptr workarea xaltintercept xaxislabel xintercept xlabels xlogscale xmajorticks xminorticks xscale yaltaxislabel yaltlabels yaltlogscale yaltmajorticks yaltminorticks yaltscale yaxislabel yintercept ylabels ylogscale ymajorticks yminorticks yscale

Methods

Chartalttopoint Charttopoint Click Clienttoscreen Close Copy Create Delete Draw Focus Hide New Open Paint Pointtochart Pointtochartalt Print Resize Save Screentoclient Send Set Show Trigger

Callbacks

onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave
onDragOver onDragStart onFocus onHide onKeyDown onKeyPress onKeyUp onMouseDown
onMouseMove onMouseUp onOpen onSend onShow onUnFocus

Check

Description

The Check class implements the check boxes, for selecting several independent on/off options. You typically set the `caption` property which indicates the caption associated with the control, and you might set the `enabled` property to 1 or 0 depending on whether a particular option is valid. The `value` property (1 or 0) represents whether the check box is selected.

For Check boxes, it is not always necessary to have any callbacks defined, since when the dialog ends or a Button is pressed you can determine the state of the Check objects using the `value` property. In other cases, use of an `onClick` callback to detect changes in the state of these controls may be preferable, and it is necessary if other items in a dialog need to depend on the state of the Check box.

To provide help text when the user moves the mouse pointer over the Check box and pauses, you can use the `tooltip` property.

Example

```

▽DEMO_Check;DEMO
[1]  ⍺ Sample function demonstrating use of the Check object
[2]  DEMO←'□' □NEW 'Dialog' ♦ DEMO.scale←1
[3]  DEMO.title←'Check Example'
[4]  ⍺
[5]  DEMO.CHECK1.New 'Check'
[6]  DEMO.CHECK1.where←2 3 1 30
[7]  DEMO.CHECK1.caption←'I run APLX on MacOS'
[8]  ⍺
[9]  DEMO.CHECK2.New 'Check'
[10] DEMO.CHECK2.where←3.5 3 1 30
[11] DEMO.CHECK2.caption←'I use a Postscript printer'
[12] ⍺
[13] DEMO.CHECK3.New 'Check'
[14] DEMO.CHECK3.where←5 3 1 30
[15] DEMO.CHECK3.caption←'I upgraded from APL.68000'
[16] DEMO.CHECK3.value←1
[17] ⍺
[18] DEMO.CHECK4.New 'Check'
[19] DEMO.CHECK4.where←6.5 3 1 30
[20] DEMO.CHECK4.caption←'I prefer Visual Basic'
[21] DEMO.CHECK4.enabled←0
[22] ⍺
[23] ⍺ Wait for the user to close the window
[24] 0 0ρ□WE DEMO
▽

```

Properties

align anchors aquaadjust autodraw caption children class color data doublebuffered dragsource droptarget enabled events extent font handle maxsize methods minsize name opened order pointer

properties scale self size sourceformats tabstop targetformats tie tooltip units value visible where winptr

Methods

Click Clienttoscreen Close Create Delete Draw Focus Hide New Open Paint Resize Screentoclient Send Set Show Trigger

Callbacks

onClick onClose ondblclick ondestroy ondragdrop ondragend ondragenter ondragleave ondragover ondragstart onfocus onhide onkeydown onkeypress onkeyup onmousedown onmousemove onmouseup onopen onsend onshow onunfocus

ChooseColor

Alternative name: ChooseColour

Description

The ChooseColor class implements the pre-defined color-selection dialog. This is a top-level object (i.e. it should not be opened as the child of a window).

To use the dialog, you must first create it using `NEW`. (Alternatively, you can create it as a top-level object using the `New` or `Create` method of `DWI`). It will remain hidden at this stage. You can optionally set the initial color selection as the first element of the `color` property (foreground color only).

The `style` property applies to Windows only (it is ignored under MacOS and Linux). It defines whether the user can select custom colors, by selecting one of:

```
0 = Allow custom colours
1 = Open with custom colours displayed
2 = Disallow custom colors
```

You can also add the following to `style`:

```
4 = Return nearest solid colour
8 = Allow any colour, dither if necessary
```

Under Windows and Linux, you can pre-define up to 16 custom colors by setting the `custom` property, as a vector of 16 integers representing the RGB values of each color. (This property is ignored under MacOS).

To set the color which will be selected when the dialog is shown, you can assign a value to the `color` property.

When you are ready to show the dialog, call the `Show` method. This displays the modal dialog. The user can then select a color; if the Cancel button is pressed, the `Show` method returns 0. If the OK button is pressed, the `Show` method returns 1. You can then read the `color` property to retrieve the color selected by the user (only the first element is used); if this is `-1`, the user has selected an undefined custom color. Under Windows and Linux, you can also optionally read the `custom` property to retrieve the list of any custom colors which the user has defined, as a vector of up to 16 integers.

Example

```
▽DEMO_ChooseColor;DLG
[1]  A Sample function demonstrating use of the ChooseColor object
[2]  DLG←'□' NEW 'ChooseColor'
[3]  A Show the dialog. If user presses Cancel, quit
[4]  :If 1=DLG.Show
```

```
[5]     a He pressed OK.  
[6]     'Color chosen: ', #1↑DLG.color  
[7]     :EndIf  
      ▽
```

Properties

children class color custom data events methods name opened properties self style tie

Methods

Close Create Delete New Open Send Set Show Trigger

Callbacks

onClose onDestroy onOpen onSend

ChooseDir

Description

The ChooseDir class implements the pre-defined directory-selection dialog. This is a top-level object (i.e. it should not be opened as the child of a window).

To use the dialog, you must first create it using `NEW`. (Alternatively, you can create it as a top-level object using the `New` or `Create` method of `OWI`). It will remain hidden at this stage. You can optionally set the initial directory as the `directory` property. You can also set the title of the dialog as the `caption` property.

When you are ready to show the dialog, call the `Show` method. This displays the modal dialog. The user can then select a directory; if the Cancel button is pressed, the `Show` method returns 0. If the OK button is pressed, the `Show` method returns 1. You can then read the `directory` property to retrieve the directory selected by the user.

Example

```

▽DEMO_ChooseDir;DLG
[1]  ⍎ Sample function demonstrating use of the ChooseDir object
[2]  DLG←' ' NEW 'ChooseDir'
[3]  ⍎ Show the dialog. If user presses Cancel, quit
[4]  :If 1=DLG.Show
[5]    ⍎ He pressed OK.
[6]    'Directory selected: ',DLG.directory
[7]  :EndIf
▽

```

Properties

caption children class data directory events methods name opened properties self tie

Methods

Close Create Delete New Open Send Set Show Trigger

Callbacks

onClose onDestroy onOpen onSend

ChooseFont

Not implemented under MacOS

Description

The ChooseFont class implements the pre-defined font-selection dialog. This is a top-level object (i.e. it should not be opened as the child of a window).

To use the dialog, you must first create it using `NEW`. (Alternatively, you can create it as a top-level object using the `New` or `Create` method of `OWI`). It will remain hidden at this stage. You can set the initially-selected font by setting the `font` property.

The `style` property is implemented under Windows only (it is ignored under Linux). It defines the fonts which can be selected:

```
0 = Screen fonts only
1 = Printer fonts
2 = Both
```

You can also add the following to `style`:

```
4 = Include strikeout/underline/color effects
8 = Exclude OEM fonts
16 = ANSI character set fonts only (no Symbol)
32 = Fixed Pitch only
64 = Scalable only
128 = TrueType only
```

If you have included 4 in the `style` property, you can also optionally set the initial color selection as the first element of the `color` property (foreground color only).

Under Windows, if you want to restrict the range of valid font sizes, set the `range` property to be a two-element vector of the minimum and maximum size, in the current `scale` of the object. (Remember that the scale defaults to 1, 'character' units.) Setting `range` to be an empty vector means there is no restriction. (*This property is not available under Linux.*)

When you are ready to show the dialog, call the `Show` method. This displays the modal dialog. The user can then select a font; if the Cancel button is pressed, the `Show` method returns 0. If the OK button is pressed, the `Show` method returns 1. The `font` property will contain the selected font as a 4-element nested vector of (Font Face) (Size) (Style) (Character set). You can also read the `color` property to retrieve the color (if any) selected by the user (only the first element is used).

Example

```

∇DEMO_ChooseFont;∅IO;VERSION;DLG
[1]  ⌘ Sample function demonstrating use of the ChooseFont object
[2]  ⌘
[3]  ⌘ Can't run this on a Mac, because it doesn't support the
[4]  ⌘ ChooseFont object
[5]  ∅IO←1
[6]  VERSION←'∅' ∅WI 'version'
[7]  →(VERSION[2]=0)/MAC
[8]  DLG←'∅' ∅NEW 'ChooseFont' ∅ DLG.scale←3
[9]  ⌘
[10] ⌘ Say we want to offer 'effects' (color and strikeout etc)
[11] DLG.style←4
[12] ⌘
[13] ⌘ Set initial values, Arial 12-point bold, red
[14] DLG.font←('Arial' 12 1)
[15] DLG.color←255
[16] ⌘
[17] ⌘ Show the dialog. If user presses Cancel, quit
[18] :If 1=DLG.Show
[19]   ⌘ He pressed OK
[20]   'Selected font: ',∅DLG.font
[21]   'Selected color: ',∅1↑DLG.color
[22] :EndIf
[23] →0
[24] MAC:
[25] 'This demo cannot be run on the Macintosh because the'
[26] 'ChooseFont object is not supported'
∇

```

Properties

children class color data events font methods name opened properties range scale self style tie

Methods

Close Create Delete New Open Send Set Show Trigger

Callbacks

onClose onDestroy onOpen onSend

Combo

Description

The Combo class implements the Combo object, a drop-down menu (or list-box) with an optional edit field.

The `style` property defines the type of combo box, as follows:

```
0 = drop down menu with edit field
1 = simple list box (always down), with edit field
2 = drop down list, no edit field
```

Note: Under MacOS, only style 2 is currently supported. Under Linux, style 1 is not supported.

The `list` property determines the items in the menu/list box part of the control. You can set it either as a character matrix or as a character vector with embedded carriage returns. (If you read it back, it will be returned as a character matrix).

Under Windows and Linux, the `text` property contains the text of the edit field (which the user may have edited), as a character vector. The `selection` and `seltext` properties contain the selection start/end and selected text in the edit field. Under Windows only, the `limit` property allows you to limit the maximum number of characters in this text.

The `value` property is an integer containing the index into the list of the selected item, in index origin 1. It is an empty vector if there is no selection (you can also write 0 to achieve the same effect).

To provide help text when the user moves the mouse pointer over the control and pauses, you can use the `tooltip` property.

If you need to detect when the user changes the selection, add an `onChange` callback.

Example

```
▽DEMO_Combo;DEMO
[1]  ⍝ Sample function demonstrating use of the Combo object
[2]  DEMO←' ' ⍵NEW 'Dialog' ⋄ DEMO.scale←1
[3]  DEMO.title←'Combo Example'
[4]  DEMO.myCombo.New 'Combo' ⋄ DEMO.myCombo.where←2 1
[5]  DEMO.myCombo.list←⍵W ⋄ DEMO.myCombo.value←3
[6]  ⍝ Wait for the user to close the window
[7]  0 0ρ⍵WE DEMO
▽
```

Properties

align anchors aquaadjust autodraw caption children class color data doublebuffered dragsource droptarget enabled events extent font handle limit list maxsize methods minsize name opened order pointer properties scale selection self seltext size sourceformats style tabstop targetformats text tie tooltip units value visible where winptr

Methods

Click Clienttoscreen Close Create Delete Draw Focus Hide New Open Paint Resize Screentoclient Send Set Show Trigger

Callbacks

onChange onClick onClose ondblclick ondestroy ondragdrop ondragend ondragenter ondragleave ondragover ondragstart onfocus onhide onkeydown onkeypress onkeyup onmousedown onmousemove onmouseup onopen onsend onshow onunfocus

Dialog

Description

The Dialog class is a special case of the Form or Window object, with the default properties already set up for a typical dialog box.

It defaults to a non-resizable, standard dialog box border, and a dialog-box background color (typically light gray for Windows and Linux, or the color from the default Aqua theme on Mac OS X). Note that, under Linux, the Window Manager determines how the request for a particular border will be interpreted; under some Window Managers, windows are always resizable.

You can use the `wait` method to make the dialog display in modal form. This means that other windows of your application are disabled until the dialog is closed.

Example

```

    ▽DEMO_Dialog;DEMO
[1]  ⍎ Sample function demonstrating use of the Dialog object
[2]  DEMO←'□' □NEW 'Dialog' ♦ DEMO.title←'Dialog Example'
[3]  DEMO.where←4 4
[4]  DEMO.scale←1
[5]  DEMO.size←10 30
[6]  ⍎
[7]  ⍎ Wait for the user to close the window
[8]  0 0ρ□WE DEMO
    ▽

```

Properties

align anchors aquaadjust autodraw border caption children class color data doublebuffered dragsource droptarget enabled events extent handle maxsize menuimagelist methods minsize name opened pointer properties scale self size sourceformats targetformats tie units visible where winptr

Methods

Click Clienttoscreen Close Create Delete Draw Focus Hide New Open Paint Resize Screentoclient Send Set Show Trigger Wait

Callbacks

onClick onClose onDb1Click onDestroy onDragDrop onDragEnd onDragEnter onDragLeave onDragOver onDragStart onFocus onHide onMenu onMouseDown onMouseMove onMouseUp onMove onOpen onPaint onResize onSend onShow onUnFocus

Document

Description

The Document class implements a special window for displaying and editing text. It is equivalent to an ordinary Window with a built-in multi-line edit control.

The text displayed in the window is represented by the `text` property, and the font used by the `font` property. Other text-edit properties include `selection`, `seltext` and `canundo`. Related methods are `Cut`, `Clear`, `Paste` and `Undo`.

You can optionally install an `onChange` callback to detect when the user edits the text.

Example

```

    ▽DEMO_Document;DEMO
[1]  # Sample function demonstrating use of the Document object
[2]  DEMO←'□' □NEW 'Document' ♦ DEMO.scale←3
[3]  DEMO.title←'Document Example'
[4]  DEMO.Font('Arial' 12 1)
[5]  DEMO.Text 'You can edit this text (if you must).'
[6]  DEMO.where←10 10
[7]  DEMO.size←130 250
[8]  #
[9]  # Wait for the user to close the window
[10] 0 0ρ□WE DEMO
    ▽

```

Properties

align anchors aquaadjust autodraw border canundo caption children class color data doublebuffered
 dragsource droptarget enabled events extent font maxsize menuimagelist methods minsize name
 opened pointer properties scale selection self seltext size sourceformats targetformats text tie units
 visible where winptr

Methods

Clear Click Clienttoscreen Close Copy Create Cut Delete Focus Hide New Open Paint Paste Resize
 Screentoclient Send Set Show Trigger Undo Wait

Callbacks

onChange onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter
 onDragLeave onDragOver onDragStart onFocus onHide onMenu onMouseDown onMouseMove
 onMouseUp onMove onOpen onPaint onResize onSend onShow onUnFocus

Edit

Description

Where the user is expected to enter text, you typically use an Edit object. Depending on the `style` property, this can be single- or multi-line, and can optionally be read-only.

The text in an Edit object is all displayed in the same font, size, style and color. (If you want to allow multiple fonts, styles, sizes and colors, use the RichEdit control instead).

At any time, the `text` property contains the displayed text; it can be set either under program control (for example, to set up an initial default) or by the user typing. In addition, the `selection` property allows you to get or set the position of the cursor and the text selection position. If you want to restrict the maximum number of characters entered, use the `limit` property.

You can set the `font`, `color` and `enabled` properties for special effects. The `border` property for Edit texts defaults to 1, which means that a border is drawn. If you do not want this, set it to 0.

The `style` property for an edit control is the sum of a set of flags:

```

1 = Centered
2 = Right justified
4 = Multiline
16 = Vertical scroll bar
32 = Inhibit auto horizontal scroll (i.e. word wrap if multiline)
64 = Horizontal scroll bar
1024 = Show selection even when control does not have focus (Windows only)
4096 = Read only
8192 = Allow Enter chars (multi-line only)
131072 = Allow Tab characters to be input, instead of tabbing to next control
262144 = Disable unnecessary scroll bars instead of removing them (MacOS only)

```

To provide help text when the user moves the mouse pointer over the control and pauses, you can use the `tooltip` property.

The `Cut`, `Copy` and `Paste` methods can be used to cut and paste to and from the clipboard.

There are several callbacks which are often used for Edit text objects. These include:

`onClick`: Invoked when the cursor position or text selection is changed

`onChange`: Invoked when the text is modified

`onFocus`: The Edit text object has got input focus

`onUnfocus`: The object has lost focus (often used to validate user input)

onLimit: The user has attempted to enter more characters than allowed by the limit property

Example

```

▽DEMO_Edit;DEMO
[1]  A Sample function demonstrating use of the Edit object
[2]  DEMO←' ' □NEW 'Dialog' ♦ DEMO.title←'Edit Example'
[3]  DEMO.myEdit.New 'Edit' ♦ DEMO.myEdit.where←2 1
[4]  DEMO.myEdit.text←'Some sample text'
[5]  A
[6]  A Wait for the user to close the window
[7]  0 0ρ□WE DEMO
▽

```

Properties

align anchors aquaadjust autodraw border canundo caption children class color data doublebuffered
 dragsource droptarget enabled events extent font handle limit maxsize methods minsize name opened
 order pointer properties scale selection self seltext size sourceformats style tabstop targetformats text
 tie units visible where winptr

Methods

Clear Click Clienttoscreen Close Copy Create Cut Delete Draw Focus Hide New Open Paint Paste
 Resize Screenshot Send Set Show Trigger Undo

Callbacks

onChange onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter
 onDragLeave onDragOver onDragStart onFocus onHide onKeyDown onKeyPress onKeyUp
 onMouseDown onMouseMove onMouseUp onOpen onSend onShow onUnFocus

Form

Alternative name: Window

Description

The Form (or Window) object is a standard window, on which can be placed other objects (such as buttons, pictures, or edit text fields). You can determine whether or not it is resizable, has close and zoom boxes, and the appearance of the title bar, by setting the `border` property when you create the window. The basic border styles are:

0	No border
1	Standard document window, window not resizable
2	Resizable document window
3	Modal-dialog style window
4	Movable modal-dialog style window
5	Thin title-bar, non-resizable window
6	Thin title-bar, resizable window

The optional flags which you can add to these basic values are:

16	Window includes a Close box
64	Window includes a Zoom box (under MacOS) or Minimize/Maximize controls (under Windows or Linux)

You cannot include a Close box if the basic `border` type is 0, and you cannot include a Zoom box if it is 0, 5 or 6.

Note that, under Linux, the Window Manager determines how the request for a particular border will be interpreted, so the border style you ask for may not be honored. Under some Window Managers, windows are always resizable.

The default value is $2+16+64 = 82$ for a Window object.

Window properties

The other main properties of a Window which you are likely to use are:

`title`: The title which appears in the Window's title bar.

`visible`: Allows you to make the window visible or invisible

`scale`: Determines the units used to place controls in the window (inherits from the System object `scale` when the window is created).

`size`: The height and width of the window, in the window's own `scale`.

`where`: The position and size of the window, in the `scale` of the `System` object.

`maxsize`: Allows you to specify the maximum size of the window

`minsize`: Allows you to specify the minimum size of the window

`bitmap`: Read-only property which returns a matrix of the pixels which make up the displayed window's contents. (*Not implemented under Linux. Under MacOS 9, valid only if the window is completely visible on screen.*)

Window methods

Methods which are useful include:

`Open`: Open a window which you have previously closed

`Close`: Close a window, either conditionally or unconditionally

`Wait`: Makes the window modal (see below)

`Draw`: Draws geometric shapes and text on the window

Window events

The event handler (callback) properties which are particularly relevant to `Window` objects are:

`onResize`: Invoked when the user changes the window's size

`onFocus`: Invoked when the window gets focus (is activated)

`onClose`: Invoked when the user wants to close the window

`onDestroy`: Invoked when the window is destroyed

Creating and Opening of Windows

In order to avoid lots of drawing and moving while creating a window and its child objects, it is always initially created in a hidden state. It is shown (unless explicitly hidden) on the first of the following:

- APL requests input
- You call `□WE`, or (under MacOS) one of the APLX library functions `GETEVENT`, `WAITEVENT` or `GETEVENTS`
- You create another window
- You explicitly open it using the `Open` or `Show` methods

Using windows as modal dialogs

The `Wait` method makes the window display in modal form, i.e. other windows of your application are disabled until the window has closed. For modal dialogs, you should consider using the `Dialog` object, which is a special case of the `Form` object in which the borders and background automatically take on the appropriate appearance for a modal dialog on the system on which APLX is running.

Deleting windows

In order to free up memory used by an object, you must ensure it is deleted when you have finished with it. If you created it using `NEW`, it will be deleted automatically when the last reference to it is erased from the workspace. (In the example below, this will happen automatically when the function ends, because the variable `DEMO` which hold the reference to the window is localized).

If you created the window using `WT`, there is no explicit reference to the window held in the workspace, so it will not be deleted automatically until the APLX session ends. You therefore need to delete it explicitly, using the `Delete` method (note that this is not the same as closing the object). You normally only need to do this for the window itself, since all child objects on the window are automatically deleted when you delete the parent.

As a convenience, windows which have no `onClose` callback defined are automatically deleted if the user closes the window. If you still have a reference to the window in the workspace because you created it using `NEW`, the reference will become a reference to an 'Unknown object'.

Example

```

▽DEMO_Form;DEMO
[1]  ⍝ Sample function demonstrating use of the Form object
[2]  ⍝ Set border to 5 (for thin title) plus 16 (Close box)
[3]  DEMO←'□' NEW 'Form' ⋄ DEMO.border←(5+16)
[4]  DEMO.title←'Form Example'
[5]  DEMO.where←4 4
[6]  ⍝
[7]  ⍝ Wait for the user to close the window
[8]  0 0ρWE DEMO
▽

```

Properties

align anchors aquaadjust autodraw bitmap border caption children class color data doublebuffered
 dragsource droptarget enabled events extent handle maxsize menuimagelist methods minsize name
 opened pointer properties scale self size sourceformats targetformats tie units visible where winptr

Methods

Click Clienttoscreen Close Create Delete Draw Focus Hide New Open Paint Resize Screenshot
 Send Set Show Trigger Wait

Callbacks

onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave
onDragOver onDragStart onFocus onHide onMenu onMouseDown onMouseMove onMouseUp
onMove onOpen onPaint onResize onSend onShow onUnFocus

Frame

Description

The `Frame` class implements a frame or group box in which you can create other controls as children. This is typically used in dialogs to group together a set of controls. For example, a set of radio buttons defined in a `Frame` object form a single group in which only one radio button can be selected at a time; a different set of radio buttons in a different `Frame` object will form a separate, independent group.

You can also use a `Frame` to contain a set of controls, such as `Buttons` or `ToolButtons`, which you want to move together on the window (for example, so they act as a toolbar which can be positioned along the top or bottom of the window).

Whether the group box title (`caption` property) is shown depends on the `style` property. If `style` is 0, the caption is shown. If `style` is 1, it is not.

If you want to display geometric shapes and draw text on the frame, you can use the `Draw` method.

Example

```

▽DEMO_Frame;DEMO
[1]  ⍝ Sample function demonstrating use of the Frame object
[2]  DEMO←'□' □NEW 'Dialog' ◇ DEMO.scale←1 ◇ DEMO.size←18 24
[3]  DEMO.title←'Frame Example'
[4]  ⍝ Create a frame with a caption and some radio buttons
[5]  DEMO.Frame1.New 'Frame' ◇ DEMO.Frame1.where←5 1 8 22
[6]  DEMO.Frame1.caption←'Radio buttons'
[7]  DEMO.Frame1.R1.New 'Radio'
[8]  DEMO.Frame1.R1.caption←'Option 1' ◇ DEMO.Frame1.R1.where←1.5 1 2 20
[9]  DEMO.Frame1.R2.New 'Radio'
[10] DEMO.Frame1.R2.caption←'Option 2' ◇ DEMO.Frame1.R2.where←3.5 1 2 20
[11] DEMO.Frame1.R3.New 'Radio'
[12] DEMO.Frame1.R3.caption←'Option 3' ◇ DEMO.Frame1.R3.where←5.5 1 2 20
[13] ⍝ Create a second frame with no caption, and some action buttons
[14] ⍝ which will align it to top or bottom of window
[15] DEMO.Frame2.New 'Frame'
[16] DEMO.Frame2.style←1 ◇ DEMO.Frame2.size←4 0 ◇ DEMO.Frame2.align←1
[17] DEMO.Frame2.B1.New 'Button'
[18] DEMO.Frame2.B1.caption←'Top' ◇ DEMO.Frame2.B1.where←1 1 2 10
[19] DEMO.Frame2.B1.onClick←"DEMO.Frame2.align←1"
[20] DEMO.Frame2.B2.New 'Button'
[21] DEMO.Frame2.B2.caption←'Bottom' ◇ DEMO.Frame2.B2.where←1 12 2 10
[22] DEMO.Frame2.B2.onClick←"DEMO.Frame2.align←3"
[23] ⍝
[24] ⍝ Wait for the user to close the window
[25] 0 0□WE DEMO
▽

```

Properties

align anchors aquaadjust autodraw caption children class color data doublebuffered dragsource droptarget enabled events extent font handle maxsize methods minsize name opened order pointer properties scale self size sourceformats style tabstop targetformats tie units visible where winptr

Methods

Click Clienttoscreen Close Create Delete Draw Focus Hide New Open Paint Resize Screentoclient Send Set Show Trigger

Callbacks

onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave onDragOver onDragStart onFocus onHide onKeyDown onKeyPress onKeyUp onMouseDown onMouseMove onMouseUp onOpen onSend onShow onUnFocus

GetMail

Description

The `GetMail` object allows your APLX applications to retrieve and manipulate e-mail messages on a mail server, using the POP3 protocol supported by most Internet Service Providers (ISPs) for reading e-mails. (A more complex protocol, IMAP, is sometimes used instead, but is not currently supported by APLX.) It complements the `SendMail` object, which can be used to send e-mails from your APLX application.

You can use the `GetMail` object for a number of purposes. For example, you might have an APLX application for which you set up a special mailbox, to which data is sent in e-mail form, analysed, and the reply sent back automatically using the `SendMail` object. Or you might use it to help process and organise your standard personal e-mails in a way which is more sophisticated than that provided by the filtering and scripting rules provided by ordinary mail programs such as Microsoft's Outlook Express; this might include automatically deleting certain unwanted e-mails *before* they are downloaded from the server.

Connecting to the mail server

The `GetMail` object should be created as a top-level object, i.e. not as the child of a Window or other control. You then need set up the following properties, to define the address of the POP3 mail server you are using, your username, and your password (all this information should be available from your ISP or network administrator):

`host`: This is a character vector, to which you should assign the Internet address of the POP3 server. It can be specified either as a domain name (such as `'pop3.myisp.com'`), or directly as an internet node address (such as `'168.9.211.53'`).

`port`: You can use this property to change the port used by the POP3 connection. For most applications you should leave this set to the default value of 110.

`user`: The user name for the account should be assigned to this property.

`password`: The password for the account should be assigned to this property.

You will probably also want to set up the `path` property. This is a character vector specifying the directory where APLX should place any attachments associated with messages you download.

Next, you need to call the `Open` method, which establishes the connection with the server. (*Note*: Whilst you have this connection open, the mailbox will be locked. You should aim to close the connection as soon as you can; if you fail to do so, the POP3 server will eventually time out, close the connection, and abort any pending requests to delete messages). The `Open` method returns an integer

scalar, which indicates whether the connection succeeded. The returned value will be one of 0 = OK, 1 = Could not find host, 2 = Authentication error (i.e. incorrect username or password), 3 = Other error.

For example:

```
Mail1←'□' □NEW 'GetMail'
Mail1.host←'pop3.supernet.com'
Mail1.user←'microapl'
Mail1.password←'secret'
Mail1.Open
0
```

Retrieving information about available messages

Once you are successfully connected to the POP3 server, the following read-only properties give information about the messages available for download:

`count`: The count of messages on the server, as an integer scalar.

`messages`: Returns a nested matrix describing the messages on the server, with one row per message. The columns (in Index Origin 1) are:

1. The 'From' field, i.e. the sender of the message, as a character vector.
2. The 'To' field, i.e. the recipient or recipients of the message, as a character vector. If there are more than one, they will be delimited by commas.
3. The 'Date' field, as a character vector, in the format 'Fri, 14 Jan 2005 10:31:00 +0000'.
4. The 'Subject' field, as a character vector.
5. The Size of the message in bytes, as an integer.
6. A Boolean scalar indicating whether the message has multiple parts.
7. The unique Message ID, as a character vector.

The messages are listed in order, so you can retrieve them by index position using the `GetMessage` or `GetSummary` methods. Message indices always start at 1, so the e-mail associated with the first row of the `messages` matrix can be retrieved as message number 1.

Retrieving individual messages or headers

Whilst you have the POP3 connection open, you can retrieve the full contents of an individual message (into properties of the `GetMail` object) by calling the `GetMessage` method. Alternatively, you can call the `GetSummary` method which retrieves the message headers only, and thus executes much faster, especially if the message is long or has large attachments. Both methods take an integer scalar argument, which is the index (in origin 1) of the message you want to retrieve. (This will be a number in the range 1 to the value of the `count` property, or the number of rows in the `messages` property).

Having successfully called `GetMessage` or `GetSummary`, you can then access the following read-only properties, which will now contain details of the retrieved e-mail message:

Valid after calling either `GetMessage` or `GetSummary`:

`subject`: The subject of the e-mail, as a character vector.

`from`: The e-mail address of the sender of the e-mail, as a character vector.

`id`: The ID of the e-mail, as a character vector. This can be used to identify the e-mail. If the Internet mailing standards have been followed correctly, this should be unique to this particular e-mail message.

`header`: The raw header of the e-mail, formatted as a character vector, with embedded carriage return (␣) characters separating lines. Each item in the header comprises a keyword, ending in a colon, followed by a text string which is the value associated with the keyword. See the description of the `header` property for more details.

`size`: An integer scalar giving the size of the e-mail in bytes.

Valid after calling `GetMessage` only:

`to`: The recipients of the e-mail, specified as a comma-delimited list of e-mail addresses.

`cc`: The list of e-mail addresses to which the message is copied, specified as a comma-delimited list.

`replyto`: The e-mail address to which any reply to an e-mail should be sent. It is a simple character vector.

`date`: The timestamp of an e-mail, as a character vector. It is normally formatted in the form 'Tue, 16 Nov 2004 12:17:40 +0000', where the last part is the offset from Greenwich Mean Time as HHMM.

`body`: The main text of the e-mail in plain text, as a simple character vector with carriage return (␣) characters separating paragraphs. If the message has been correctly formatted by the program which sent it, this property should always contain the plain-text version of the message. However, some e-mail applications, and many senders of 'spam' (junk) mail, place HTML text here.

`html`: The main text of the e-mail as HTML, if the message as sent in HTML form. If the message was sent in plain-text form only, this property will be an empty vector. (*Note:* You can write the HTML text to a temporary file and use the `Browser` object to display it.)

`attachments`: A list of file names which correspond to the attachments of the mail message, as a nested vector of file names. The `GetMessage` method will have retrieved the attachments and written them out to these files. They will be placed in the directory specified by the `path` property. If there is a name clash, i.e. an attachment has the same name as an existing file, APLX will append .1, .2 etc to distinguish the different files. If for some reason the attachment cannot be written (for example, if the

path property is invalid, or the disk is full), APLX will report a `FILE I/O ERROR` when you run the `GetMessage` method.

For example, assuming you have a reference to a `GetMail` object in `Mail1` and have successfully connected to the server, the following function would fetch the `N`th e-mail from the server, and display the message in the Session window. Any attachments would also be retrieved and saved to local files.

```

▽GetMail N
[1]  ⍝ Get and display message N
[2]  Mail1.GetMessage N
[3]  'To:           ',Mail1.to
[4]  'From:         ',Mail1.from
[5]  'CC:           ',Mail1.cc
[6]  'Subject:      ',Mail1.subject
[7]  'Attachments: ',⍝Mail1.attachments
[8]  50p'- '
[9]  Mail1.body
[10] 50p'- '
▽

```

Caution: Before calling the `GetMessage` method, consider whether you want the message to be deleted, or left on the server, as described in the next paragraph.

Deleting messages on the server

The `DeleteMessage` method marks a message on the server for deletion. It takes an integer scalar argument, which is the index number of the message to be deleted.

Alternatively, you can set the `deleteonread` property of the `GetMail` object to 1. This causes messages which are retrieved using the `GetMessage` method to be marked for deletion automatically.

The server will not actually delete the messages until the session is successfully completed and you call the `Close` method. If an error occurs, or if the connection is lost, the message will be left untouched. They will also not be deleted if you call the `Reset` method.

Closing the connection

Finally, you should call the `Close` method to terminate the POP3 session. This releases the mailbox lock, and causes the server to remove any messages which have been marked for deletion.

Error reporting and diagnostics

If an error occurs when calling the `GetMessage` or `GetSummary` methods, an `APL I/O ERROR` will be generated. If an error occurs when the connection is being established using the `Open` method, it will return a non-zero result, as described above. In either case, you can use the `status` property to find out more about the error. This returns a two-element integer vector. The first element is 1 if the connection to the POP3 server is active, else 0. The second element is the latest error code returned by the underlying operating-system networking code. (See for example, the Windows documentation and include file 'winsock.h' for details on these error codes.)

Another useful diagnostic tool is the `serverreply` property. This returns a character vector containing the status message which the server sent back after a call such as `GetMessage` or `Open`. Usually it will begin "+OK..." to indicate success, but if there is a problem (for example, if the mailbox is already locked) it will contain an error message. This can comprise several lines, separated by carriage returns.

Other properties and methods

`Reset`: This is a method which takes no arguments. It resets the connection to the server. Any pending deletes are cancelled.

`timeout`: This is an integer scalar property, which allows you to set the timeout for mail operations, in milliseconds.

Example

```

vDEMO_GetMail;host;user;password;errcode;messages;count;IO;GM;W
[1]  A Sample function demonstrating use of the GetMail object
[2]  A To run this function, you need an Internet account with
[3]  A a POP3 mail server, and you need to know the host name,
[4]  A user name and password for this account.
[5]  A
[6]  A Ask the user for the mail account details
[7]  M←'Enter mail server (POP3) host name: ' ♦ host←DDBRM
[8]  M←'Enter user name:                    ' ♦ user←DDBRM
[9]  M←'Enter password:                    ' ♦ password←DDBRM
[10] A
[11] A Create the GetMail object and set up connection details
[12] A Don't delete messages on the server when we read them
[13] GM←'M' MNEW 'GetMail' ♦ GM.deleteonread←0
[14] GM.host←host ♦ GM.user←user ♦ GM.password←password
[15] A
[16] A Open the connection to the POP3 server
[17] errcode←GM.Open
[18] :If errcode≠0
[19]   :Select errcode
[20]   :Case 1
[21]     'Could not find host'
[22]   :Case 2
[23]     'Authentication error'
[24]   :Else
[25]     'Comms error, error code = ',#1↓GM.status
[26]   :EndSelect
[27]   :Return
[28] :EndIf
[29] A
[30] A Get list of messages and display them in a Grid object
[31] IO←1
[32] count←GM.count
[33] :If count>0
[34]   'Fetching list of messages from server...'
[35]   messages←GM.messages
[36]   A Close connection as soon as possible
[37]   A In a real application, you would fetch or delete selected messages here
[38]   GM.Close
[39]   A
[40]   W←'M' MNEW 'Window' ♦ W.caption←'Messages' ♦ W.scale←5 ♦ W.size←500 430
[41]   W.Show

```

```

[42] W.Grid.New 'Grid' ◊ W.Grid.align←1
[43] W.Grid.headcols←0 ◊ W.Grid.rows←count ◊ W.Grid.cols←5
[44] W.Grid.colsize←(1p0)150
[45] W.Grid.text←(1)(15)('From' 'To' 'Date' 'Subject' 'Size')
[46] W.Grid.value←(1count)(15)(messages[;15])
[47]  A Wait for the user to close the window
[48]  0 0pWE W
[49] :Else
[50] 'You have no messages available'
[51]  A Close connection
[52]  GM.Close
[53] :EndIf
[54]  A

```

Properties

attachments body cc children class count data date deleteonread events from header host html id messages methods name opened password path port properties replyto self size status subject tie timeout to user

Methods

Close Create Delete Deletemessage GetMessage Getsummary New Open Reset Send Set Trigger

Callbacks

onClose onDestroy onOpen onSend

Licensing (*Windows and Linux versions*)

In the Windows and Linux versions of APLX, the GetMail object is based on the Indy networking classes. The following notices apply to these versions:



Portions of this software are Copyright (c) 1993 - 2003, Chad Z. Hower (Kudzu) and the Indy Pit Crew - <http://www.IndyProject.org/>.

THIS SOFTWARE IS PROVIDED BY Chad Z. Hower (Kudzu) and the Indy Pit Crew "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Grid

Introduction

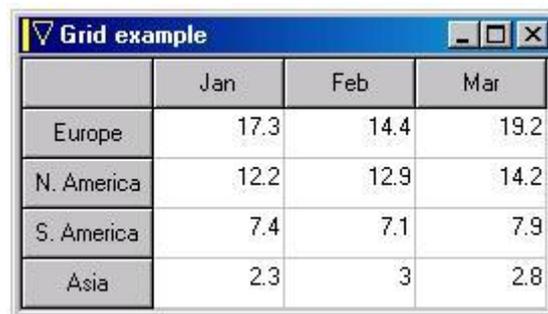
The Grid object allows the display, optionally with editing, of two-dimensional data in a form similar to that of a spreadsheet. By placing Grid objects on a Page object within a tabbed Selector, you can add further dimensions to the data.

The Grid is made up of a rectangular array of *cells*, which can contain either text or numeric data. At the top and left of the grid you can optionally have header (non-scrolling) cells.

In this example, we create a grid with 4 rows and 3 columns of scrolling (data) cells, and one row and column of header cells:

```
W←'□' □NEW 'Window' ◇ W.caption←'Grid example'
W.Grid.New 'Grid' ◇ W.Grid.align←¯1 ◇ W.Grid.rows←4 ◇ W.Grid.cols←3
W.Grid.text (¯1) (ι3) ('Jan' 'Feb' 'Mar')
W.Grid.text (ι4) (¯1) ('Europe' 'N. America' 'S. America' 'Asia')
DATA←4 3p17.3 14.4 19.2 12.2 12.9 14.2 7.4 7.1 7.9 2.3 3 2.8
W.Grid.value (ι4) (ι3) DATA
```

This produces the following effect:



	Jan	Feb	Mar
Europe	17.3	14.4	19.2
N. America	12.2	12.9	14.2
S. America	7.4	7.1	7.9
Asia	2.3	3	2.8

Cell Properties

As well as ordinary properties, a Grid object has properties which are associated with individual cells, known as *cell properties*. In the example above, we have used the `text` cell property to set the text within the header cells. We have also used the `value` cell property to set the contents of the data (scrolling) cells, in this case to be numbers. Cell properties are set with the special dot-notation syntax (rather as though they were methods):

```
WindowRef.ControlName.PropertyName Rows Cols Array
```

where `Rows` is a scalar or vector of the row indices you want to change, `Cols` is a scalar or vector of the column indices you want to change, and `Array` is an array of the data you want to assign to the cells. As with normal APL indexed assignment, the shape of `Array` must match the number of rows and columns specified, or be a scalar (in which case the data is assigned to all the cells specified). If

either Rows or Cols has only one element, a vector can be used for Array instead of a one-row or one-column matrix.

If you are using the older `⊞WI` syntax, the equivalent is:

```
'WindowName.ControlName' ⊞WI 'PropertyName' Rows Cols Array
```

The syntax for reading back cell properties is similar, simply omitting the Array parameter:

```
R←WindowRef.ControlName.PropertyName Rows Cols
```

or the equivalent

```
R←'WindowName.ControlName' ⊞WI 'PropertyName' Rows Cols
```

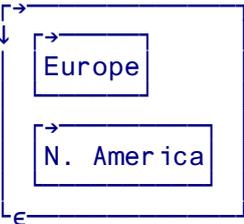
The result will be an array of shape $(\rho\text{Rows}) (\rho\text{Cols})$

Rows and columns in the data (scrolling) region are numbered starting from 1. Rows and columns in the header (non-scrolling) region are numbered $\bar{1}$, $\bar{2}$ etc, where $\bar{1}$ means the row or column nearest to the data area. In some cases, you can specify a row or column of 0, which means 'apply to the whole row/column'.

Using our example above:

```
W.Grid.text (1 2) ( $\bar{1}$ )
Europe
N. America

⊞DISPLAY W.Grid.text (1 2) ( $\bar{1}$ )
```



```
W.Grid.value (1 2) (1 3)
17.3 19.2
12.2 14.2
```

or, using `⊞WI` syntax, something like this (assuming `Win` is the name of the window):

```
'Win.Grid' ⊞WI 'text' (1 2) ( $\bar{1}$ )
Europe
N. America
'Win.Grid' ⊞WI 'value' (1 2) (1 3)
17.3 19.2
12.2 14.2
```

Numeric versus Text cells

Each cell can be of type Text or Numeric. Text cells contain simply character vectors; when you read either the `text` or `value` property of the cell, you get the text shown in the cell, possibly edited by the user, as a character vector.

Numeric cells contain numbers. If you read the `value` property of a Numeric cell, you get a numeric scalar, which is the current value of the cell. If the cell does not currently contain a valid number (because the user has edited it), you get a special numeric value to indicate an invalid number (by default this is a very large negative number; you can change it using the `conversionerrorvalue` property). If you read the `text` property of the cell, you get the actual text shown in the cell, which is usually the text representation of the cell's value, but may be different if it has been edited by the user and is not a valid value. You can also read the `valid` cell property, which tells you whether the cell contains a valid value.

With our example above:

```

W.Grid.valid (1 2) (1 3)
1 1
1 1

□□DISPLAY W.Grid.value (1 2) (1 3)
┌───┐
│17.3 19.2│
│12.2 14.2│
└───┘

□□DISPLAY W.Grid.text (1 2) (1 3)
┌───┐
│┌───┐┌───┐│
│17.3 19.2│
│┌───┐┌───┐│
│12.2 14.2│
└───┘

```

In this example, the `text` property returns a 2 by 2 nested array of character vectors, but the `value` property returns a 2 by 2 numeric matrix.

Initially, all cells default to type Text. A cell becomes of type Numeric if you write a numeric value to its `value` property, or if you write a format string to its `format` property, which determines how it is formatted in the cell.

Properties of the Grid object

style: (Integer) The `style` property is the sum of a set of flags which determine how the Grid behaves:

- 1 If set, user can select a range of cells (Default: Off)
To allow the mouse to be used to select the range,
you should also set `autoeditstart` to 0.

- 2 If set, a whole row is selected together (Default: Off)
- 4 If set, user can use the Tab key to move between cells (Default: On)
- 8 Reserved
- 16 If set, vertical scroll bar is shown if the Grid's contents will not fit between the top and bottom of the visible area (Default: On)
- 32 Reserved
- 64 If set, horizontal scrollbar is shown if the Grid's contents will not fit between the left and right of the visible area (Default: On)
- 128 If set, user can re-size rows (Default: Off)
- 256 If set, user can re-size columns (Default: On)
- 512 If set, user can move rows to re-arrange their order (Default: Off)
- 1024 If set, user can move columns to re-arrange their order (Default: Off)
- 2048 If set, the contents of the grid update as the scroll-bar is moved (Default: Off. Under MacOS, this flag is ignored since the grid always updates immediately).
- 4096 If set, unnecessary scroll bars are disabled instead of being hidden (MacOS only)

borderstyle: (Integer) The `borderstyle` property is the sum of a set of flags which determine the appearance of the Grid (the defaults are all On):

- 1 Header cells have separator horizontal line
- 2 Header cells have separator vertical line
- 4 Data cells have separator horizontal line
- 8 Data cells have separator vertical line
- 16 Use 3D effect (Windows and MacOS only, ignored under Linux)

autoeditstart: (Integer) Determines whether the data in the Grid can be edited, and if so how the edit process starts. It can be one of:

- 0 Editing of a cell is possible only when the program calls `Editstart`
- 1 Editing is automatically enabled when the cell is selected; the user can immediately type
- 2 The user must press Enter (or, under Windows and Linux, F2) to start editing a cell

The default is 1.

color or **colour**: (1, 2, 3 or 6 element numeric) The `color` or `colour` property for a Grid is set in the same way as for other controls. The foreground color you set determines the default color of the text shown in the Grid (you can change this for individual cells using the `colortext` cell property). The background color is used as the background for the data part of the Grid (you can change this for individual cells using the `colorback` cell property).

colorhead or **colourhead**: (1 or 3 element numeric) The `colorhead` or `colourhead` property for a Grid can be set as either a single RGB encoded integer (256 = Blue Green Red) or as three Red Green Blue values, each 0 to 255. It determines the background color used for header cells. The special value `-1` means use the Grid control's foreground color. The special value `-2` means use the default (light gray).

font: (Character vector or nested vector) The `font` for a Grid is set in the same way as for other controls. It determines the default font for the text shown in the Grid. You can change the font style for individual cells using the `fontstyle` cell property.

conversionerrorvalue: (Floating-point number) Value to return if a numeric cell cannot be converted to valid number. The default is a large negative number.

gridlines: (Boolean) Determines whether the grid lines within the data area are visible. (Default 1)

headcols: (Integer) The number of heading (fixed) columns (Default 1). You typically set this property when you create the grid (before setting the `cols` property).

headrows: (Integer) The number of heading (fixed) rows (Default 1). You typically set this property when you create the grid (before setting the `rows` property).

cols: (Integer) The number of data (scrolling) columns. You typically set this property when you create the Grid to fit the size of the data you want to display.

rows: (Integer) The number of data (scrolling) rows. You typically set this property when you create the Grid to fit the size of the data you want to display.

col: (Integer) The Column number (in index origin 1) of the currently-active cell. You can write to this property to change the current cell.

row: (Integer) The Row number (in index origin 1) of the currently-active cell. You can write to this property to change the current cell.

activecell: (2-element integer vector) The Row and Column of the currently-active cell. You can write to this property to change the current cell.

text: (Character vector) Although `text` is a cell property for a Grid object, as a convenience you can read (but not write to) `text` as a simple property of the Grid object. It returns the text of the currently-active cell.

value: (Character vector or numeric scalar) Although `value` is a cell property, as a convenience you can read (but not write to) `value` as a simple property of the Grid object. It returns the value of the currently-active cell. If the cell is a text cell, it will return a character vector. If it is a numeric cell, it will return the numeric value of the cell, or (if the cell is invalid) the current `conversionerrorvalue`.

selection: (4-element integer vector) The Row and Column of the top-left of the selected cell range, followed by the number of rows selected and the number of columns selected. If the `style` property is set so that only single-selection is possible (which is the default), the last two elements will both be one. You can write to this property to change the selection under program control.

view: (4-element integer vector) Determines the visible portion of the grid. The four elements are the First visible row, First visible column, Number of visible Rows, Number of visible columns. You can write to this property to scroll the grid under program control (only the first two elements are required).

firstvisible: (2-element integer vector) Determines the top-left position of the visible portion of the Grid. It is the same as the first two elements of the `view` property.

imagesize: (Read-only, 2-element numeric vector) Returns the total height and width, in the control's current scale, of all the cells in the Grid. This may be smaller than or larger than the height and width of the visible Grid control itself.

Cell properties of the Grid object

These are all set using the syntax:

```
WindowRef.ControlName.PropertyName Rows Cols Array
```

and read using the syntax:

```
R←WindowRef.ControlName.PropertyName Rows Cols
```

value: (Each cell element is a character vector or numeric scalar) Read or set cell values.

When you read this cell property for a Grid object, it returns a character vector for each Text cell and a numeric scalar for each Numeric cell, for each cell in the list of rows and columns specified. If a Numeric cell is invalid, it will return the current `conversionerrorvalue`. If you read a set of cells some of which are Numeric and some Text, it will return a nested array containing a mixture of character vectors and numeric scalars. If you read a set of cells all of which are Numeric, it will return a simple numeric array.

When you write to this cell property for a Grid object, it sets the value for each cell in the list of rows and columns specified. If you supply a number for a cell, that cell becomes a Numeric cell. If you supply a character vector or scalar, it becomes a Text cell. Writing to a cell also causes its `valid` property to be set to 1.

text: (Each cell element is a character vector) Read or set cell text.

When you read this property, it returns a character vector containing the text displayed in the cell, irrespective of the cell's type, for each cell in the list of rows and columns specified. For multiple cells, it will therefore return a nested array of character vectors.

When you write to this property, the text displayed in the cell is set to the text you supply. To set multiple cells in one operation, provide a nested array of character vectors. The type of the cell, and the cell's `valid` property, are not changed. If the cell is Numeric, it will remain so, and its `value` will not change.

Note: The text in the cell can contain embedded carriage-return (␣R) characters, which results in the text being displayed on multiple lines. In this case, you may need to increase the row height.

valid: (Each cell element is a Boolean scalar, read-only) Returns 1 if the cell is valid, and 0 if it is invalid. Text cells are always valid. Numeric cells may be invalid if the user has edited the text.

allowselection: (Each cell element is a Boolean scalar) If this is set to 1 (default), the cell can be selected. If it is set to 0, the cell cannot be selected.

textalign: (Each cell element is an integer scalar) Determines how the text is aligned within the cell. The value is the sum of two numbers. The horizontal alignment is 1 for left, 2 for right, or 4 for center. The vertical alignment is 8 for top, 16 for bottom, or 32 for center. The default value depends on the type of the cell. Header cells default to centered in both directions (4 + 32 = 36). Text cells in the data area default to top, left (1 + 8 = 9). Numeric cells default to top, right (2 + 8 = 10).

colortext or **colourtext:** (Each cell element is an integer scalar) Determines the color of text in the cell. The value is a color expressed as a numeric scalar (256 ⊂ Blue Green Red). The special value of `-1` means use the Grid foreground color for this cell; this is the default.

colorback or **colourback:** (Each cell element is an integer scalar) Determines the background color for the cell (when it is not selected). The value is a color expressed as a numeric scalar (256 ⊂ Blue Green Red). The special value of `-1` means use the Grid background color for this cell; this is the default.

fontstyle: (Each cell element is an integer scalar) Determines the style of the text within the cell. The value is the sum of:

0	Plain
1	Bold
2	Italic
4	Underlined
8	Hollow (supported under MacOS only)
16	Strikeout (not supported under MacOS)

The special value of `-1` means use the default font style of the Grid.

format: (Each cell element is a character vector) Determines the format used to display numeric values. You can specify separate formats for positive, negative and zero numbers. See the description of the `format` cell property for details of the format strings.

Setting this value automatically sets the cell type to Numeric.

Row and Column sizes

You can set the size of the rows and columns using the `rowsize` and `colsize` properties, as follows:

```
WindowRef.ControlName.rowsize Rows Sizes
WindowRef.ControlName.colsize Cols Sizes
```

or:

```
'WindowName.ControlName' □WI 'rowsize' Rows Sizes
'WindowName.ControlName' □WI 'colsize' Cols Sizes
```

where Rows or Cols is a scalar or vector list of row/column numbers, and Sizes is a matching vector (or scalar) of the row/column sizes in pixels. A row or column number of 0 sets the default size for all rows or columns which are not explicitly set. *Note: At least one of Rows/Cols and Sizes parameters must be a vector, otherwise the statement will be interpreted as an attempt to read back the current sizes.*

You can read back the current sizes using the syntax:

```
R ← WindowRef.ControlName.rowsize Rows
R ← WindowRef.ControlName.colsize Cols
```

or:

```
R ← 'WindowName.ControlName' □WI 'rowsize' Rows
R ← 'WindowName.ControlName' □WI 'colsize' Cols
```

Note that, depending on the flags you have set in the `style` property, the user may be able to resize rows and columns.

Methods for the Grid object

EditStart: This method takes no arguments. It initiates an editing session on the currently-active cell. It is useful mainly if you have set the `autoeditstart` property to 0. For example, you might initiate an edit session for certain cells only (in response to an `onSelChange` event).

DeleteCols and **DeleteRows:** These methods take an argument which is a vector of rows or columns to be deleted. After they have been deleted, the remaining rows or columns are re-numbered to form an integer sequence starting at 1.

InsertCols and **InsertRows:** These methods take an argument which is a vector of positions at which you wish to insert new rows or columns. You can specify integers (for example 5 means 'insert before the current row/column 5'), or non-integers (for example 4.5 means 'insert between the current row/column 4 and 5'). After the insertion, the rows or columns are re-numbered to form an integer sequence starting at 1. Thus, if you insert at 5 and 6, the new rows/columns become numbers 5 and 7. To insert three new rows/columns before the current second one, specify 2 2 2.

PointtoCell: Converts a point on the visible Grid control to cell row/column.

Callbacks for the Grid object:

onChange: This event is triggered when the user has edited the contents of a cell. The event-specific parameters in `⌈EV` are (in index origin 1):

```
⌈EV[6]    Row number of the edited cell
⌈EV[7]    Column number of the edited cell
⌈EV[8]    Flag to indicate whether the contents were valid
```

onColMoved: This event is triggered when the user has moved a column by dragging it to a different position in the grid. (This is possible only if the `style` property includes the flag 1024, column moving allowed). The event-specific parameters in `⌈EV` are (in index origin 1):

```
⌈EV[6]    Column number before the move
⌈EV[7]    The position of the column after the move
```

onRowMoved: This event is triggered when the user has moved a row by dragging it to a different position in the grid. (This is possible only if the `style` property includes the flag 512, row moving allowed). The event-specific parameters in `⌈EV` are (in index origin 1):

```
⌈EV[6]    Row number before the move
⌈EV[7]    The position of the row after the move
```

onColSized: (*MacOS only*) This event is triggered when the user has resized a column. (This is possible only if the `style` property includes the flag 256, column re-sizing allowed). The event-specific parameters in `⌈EV` are (in index origin 1):

```
⌈EV[6]    Column number
```

onRowSized: (*MacOS only*) This event is triggered when the user has resized a row. (This is possible only if the `style` property includes the flag 128, row re-sizing allowed). The event-specific parameters in `⌈EV` are (in index origin 1):

```
⌈EV[6]    Row number
```

onScroll: This event is triggered when the user has scrolled the Grid control.

onSelChange: This event is triggered when the user selects a new cell. The event-specific parameters in `⌈EV` are (in index origin 1):

```
⌈EV[6]    Row number of the newly-selected cell
⌈EV[7]    Column number of the newly-selected cell
```

Example

```
▽DEMO_Grid;DATA;SIZE;Demo
[1]  ⍺ Sample function demonstrating use of the Grid object
[2]  ⍺
[3]  ⍺ Create basic Grid control, without automatic edit
[4]  Demo←'⌈' ⌈NEW 'Window' ⌈ Demo.caption←'Grid example' ⌈ Demo.scale←5
[5]  Demo.Grid.New 'Grid' ⌈ Demo.Grid.align←1 ⌈ Demo.Grid.autoeditstart←2
[6]  Demo.Grid.rows←4 ⌈ Demo.Grid.cols←3
```

```

[7]   A
[8]   A Add headings (text cell properties)
[9]   Demo.Grid.text (-1) (-1) ('Profit')
[10]  Demo.Grid.text (-1) (13) ('Jan' 'Feb' 'Mar')
[11]  Demo.Grid.text (14) (-1) ('Europe' 'N. America' 'S. America' 'Asia')
[12]  A
[13]  A Insert data (numeric cell properties)
[14]  DATA←4 3p17.33 14.41 19.21 12.2 12.94 14.23 7.4 7.17 7.92 -2.32 -3.03 -2.85
[15]  Demo.Grid.value (14) (13) DATA
[16]  A
[17]  A Add format strings for the numbers, different for positive/negative
[18]  Demo.Grid.format (14) (13) (c'$.00M; ($.00M)')
[19]  A
[20]  A Set some foreground colors for cells
[21]  Demo.Grid.colortext (4) (13) 255
[22]  A
[23]  A Set header text styles
[24]  Demo.Grid.fontstyle (-1) (-1) 1
[25]  Demo.Grid.fontstyle (-1) (13) 2
[26]  A
[27]  A Make the window fit snugly around the Grid control, with a small margin
[28]  SIZE←Demo.Grid.imagesize
[29]  Demo.size←(SIZE+4)
[30]  A
[31]  A Wait for the user to close the window
[32]  0 0pWE Demo
      ▽

```

Properties

activecell align anchors aquaadjust autodraw autoeditstart borderstyle caption children class col color colorhead cols colsize conversionerrorvalue data doublebuffered dragsource droptarget enabled events extent firstvisible font gridlines handle headcols headrows maxsize methods minsize name opened order pointer properties row rowsize scale self size sourceformats style tabstop targetformats tie tooltip units view visible where winptr

Cell Properties

allowselection colorback colortext fontstyle format text textalign valid value

Methods

Click Clienttoscreen Close Create Delete Deletocols Deleterows Draw Editstart Focus Hide Insertcols Insertrows New Open Paint Pointtocell Resize Screenshotclient Send Set Show Trigger

Callbacks

onChange onClick onClose onColMoved onDbClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave onDragOver onDragStart onFocus onHide onKeyDown onKeyPress onKeyUp onMouseDown onMouseMove onMouseUp onOpen onRowMoved onScroll onSelChange onSend onShow onUnFocus

Icon

Description

The `Icon` class allows you to display standard fixed images in your windows. It is used for the simple icons defined for standard dialogs, by setting the `icon` property to a scalar integer as follows:

```
0 = Stop
1 = Alert
2 = Note
3 = Question
```

Under *APLX for MacOS*, you can also set the `icon` property to an arbitrary resource ID for an icon.

See also the `Draw` method which allows you to draw icons, pictures and bitmaps on your windows and controls.

Example

```

▽DEMO_Icon;DEMO
[1]  A Sample function demonstrating use of the Icon object
[2]  DEMO←'□' □NEW 'Dialog' ♦ DEMO.title←'Icon Example'
[3]  DEMO.myIcon.New 'Icon' ♦ DEMO.myIcon.where←2 1
[4]  DEMO.myIcon.icon←2
[5]  A
[6]  A Wait for the user to close the window
[7]  0 0ρ□WE DEMO
▽
```

Properties

align anchors aquaadjust autodraw caption children class color data dragsource droptarget enabled events extent icon maxsize methods minsize name opened pointer properties scale self size sourceformats targetformats tie units visible where

Methods

Click Clienttoscreen Close Create Delete Hide New Open Paint Resize Screentoclient Send Set Show Trigger

Callbacks

onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave onDragOver onDragStart onHide onMouseDown onMouseMove onMouseUp onOpen onSend onShow

HTTPClient

Description

The HTTPClient object allows your APLX applications to retrieve information over the internet using the HTTP protocol used by the World Wide Web. For example, you can use it to download the HTML text of a web page, or to download a binary file from a web site. It should be created as a top-level object, i.e. not as the child of a Window or other control.

Using the `Get` method

The HTTPClient object is very easy to use. In most cases, all you need to do is call the `Get` method to retrieve the data associated with a web address (URL). This takes as an argument a character vector containing the URL to retrieve (e.g. `'http://www.microapl.co.uk/apl/index.html'`).

The result is a three element nested vector.

The first element is an integer scalar. If the web server was successfully contacted, this will be the HTTP return code. A number in the range 200 to 299 indicates success. A number in the range 400 to 499 indicates an error, such as 404 meaning 'page not found'. (See below for a list of HTTP return codes.) If the web server could not be contacted, for example because the server name was not found, or your network connection has failed, the return code will be `-1`. In this case, the `status` property will contain more details on the underlying error.

The second element is a character vector indicating the MIME type of the returned data, for example `'image/gif'` or `'text/plain'`.

The third element is a character vector containing the data. This will usually be the HTML text of the web page (possibly with embedded carriage returns), translated to the APLX character set. For a binary file or image, or any data for which the MIME type does not begin with `'text'`, it will be the raw bytes returned by the web-server, without translation. If there is no data, or an error occurs, this will be an empty vector.

For example:

```

HTTP←'⎵' ⎵NEW 'HTTPClient'
      (errcode mimetype text) ← HTTP.Get 'http://www.microapl.co.uk/apl'
errcode
200
mimetype
text/html
ptext
10254
100↑text
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!-- $$PAGE_TITLE$$='APLX<sup><small>
```

Using the `Post` and `Head` methods

The `Post` and `Head` methods are similar to `Get`, and have the same syntax, arguments, and result structure. `Post` is used for HTML forms and pages where you need to send data to the web server; the argument has the data appended to the URL separated by a question mark, with `&` substituted for space (for example, `'http://www.blah.com?request=doit&code=101'`).

`Head` is the same as `Get`, except that it returns the HTTP header associated with the page, not the page itself. For example:

```
(errcode mimetype text) ← HTTP.Head 'http://www.microapl.co.uk/ap1'
errcode
200
  mimetype
text/html
  text
Server: Zeus/4.2
Date: Wed, 09 Feb 2005 12:19:41 GMT
Content-Type: text/html
Content-Length: 10437
Accept-Ranges: bytes
Last-Modified: Thu, 07 Oct 2004 11:35:34 GMT
```

Properties of the `HTTPClient` object

`user` and `password`: Set the user name and password for web pages which require login, as character vectors.

`style`: An integer scalar, which is the sum of the codes 1 meaning 'Allow Cookies' and 2 meaning 'Allow Redirects'. The default is 3.

`timeout`: An integer scalar, which is the timeout value in milliseconds.

`proxy`: Some corporate networks may require you to send HTTP requests via a 'proxy server'. If this is the case, you need to set the `proxy` property as a nested vector of (Proxy address) (Port number). Your network administrator will be able advise on whether this applies to your network. The default is an empty vector, meaning no proxy server is required.

`referrer`: This property contains the Referrer URL which will be passed to the web server when you access a page using the `HTTPClient` object. It is typically used by web administrators to keep track of the page which contained the link by which a given user reached the web-site. Normally you can leave this as an empty vector.

`status`: (*Read-only*) This returns a two-element integer vector. The first element is reserved for future use. The second element is the latest error code returned by the underlying operating-system networking code. (See for example, the Windows documentation and include file 'winsock.h' for details on these error codes.)

`url`: (*Read-only*) This property contains the actual Uniform Resource Location (i.e. web page address) which has just been retrieved. This may not be the same as the page you requested, because the request may have been re-directed.

`cookie`: (*Read-only*) Once you have used the `Get` method to retrieve the a page from a web-site, this property contains the text of any 'cookie' sent by the web server. It will be an empty vector if no 'cookie' was sent. Under Windows and Linux, the cookie will automatically be sent back to the web server if you access the same site again, unless you have disallowed cookies using the `style` property. Under MacOS, cookies are always disallowed.

HTTP Return Codes

Informational

- 100 Continue
- 101 Switching Protocols

Successful

- 200 OK
- 201 Created
- 202 Accepted
- 203 Non-Authoritative Information
- 204 No Content
- 205 Reset Content
- 206 Partial Content

Redirection

- 300 Multiple Choices
- 301 Moved Permanently
- 302 Moved temporarily
- 303 See Other
- 304 Not Modified
- 305 Use Proxy
- 306 (Unused)
- 307 Temporary Redirect

Client Error

- 400 Bad Request
- 401 Unauthorized
- 402 Payment Required
- 403 Forbidden
- 404 Not Found
- 405 Method Not Allowed
- 406 Not Acceptable
- 407 Proxy Authentication Required
- 408 Request Timeout

409 Conflict
 410 Gone
 411 Length Required
 412 Precondition Failed
 413 Request Entity Too Large
 414 Request-URI Too Long
 415 Unsupported Media Type
 416 Requested Range Not Satisfiable
 417 Expectation Failed

Server Error

500 Internal Server Error
 501 Not Implemented
 502 Bad Gateway
 503 Service Unavailable
 504 Gateway Timeout
 505 HTTP Version Not Supported

Example

```

    ▽DEMO_HTTPClient;errcode;mimetype;data;address;HTTP
[1]   a Sample function demonstrating use of the HTTPClient object
[2]   HTTP←' ' □NEW 'HTTPClient'
[3]   □←'Enter web address (for example, http://www.bbc.co.uk): '
[4]   address←□DBR □
[5]   (errcode mimetype data)←HTTP.Get address
[6]   :If (errcode<200)▽(errcode>300)
[7]     'An error occurred, code = ',errcode
[8]   :Else
[9]     'Got the page OK.'
[10]    'Asked for: ',address
[11]    'Actual URL: ',HTTP.url
[12]    'MIME Type: ',mimetype
[13]    'Size: ',⊗pdata
[14]    'Cookie: ',HTTP.cookie
[15]   :EndIf
    ▽
  
```

Properties

children class cookie data events methods name opened password properties proxy referrer self status style tie timeout url user

Methods

Close Create Delete Get Head New Open Post Send Set Trigger

Callbacks

onClose onDestroy onOpen onSend

Licensing (*Windows and Linux versions*)

In the Windows and Linux versions of APLX, the HTTPClient object is based on the Indy networking classes. The following notices apply to these versions:



Portions of this software are Copyright (c) 1993 - 2003, Chad Z. Hower (Kudzu) and the Indy Pit Crew - <http://www.IndyProject.org/>.

THIS SOFTWARE IS PROVIDED BY Chad Z. Hower (Kudzu) and the Indy Pit Crew "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Image

Description

This class implements a (possibly invisible) image-manipulation object, using the ImageMagick™ open-source package. It can be used to load images from file in over 90 different formats, transform them, optionally draw on them from APLX, and save them, again in a wide range of different formats.

It can also be used to manipulate an APL pixel array, i.e. a matrix of pixel values where each point is represented by an integer which encodes the Red, Green and Blue intensities. This can be created directly in APL, or can be read from the `bitmap` property of the System object (i.e. retrieved from the Clipboard), or from a Chart, Picture or Window. It can also be created using the `Draw` method `GetBitmap` keyword.

You can also use the `Draw` method to 'draw' directly on the off-screen bitmap of an Image object, allowing you to create your own image from within APLX, or add graphic elements to an existing image read from file. You can then save the image in one of the many formats supported by ImageMagick.

A single Image object can contain a sequence of actual images. These might, for example, be displayed as a moving image in an animated GIF. The Image object also allows you to combine images, for example for place a semi-transparent foreground picture on top of a background picture.

Note that the Image object itself is not visible, and cannot be placed directly on a window. Instead, it can either be created at the top level (child of the System object), or be a child of a Picture object. In the latter case, it effectively becomes visible, since whenever it is changed, the Picture object will automatically transfer the current bitmap of the Image to the Picture.

About ImageMagick

ImageMagick is copyright ImageMagick Studio LLC, a non-profit organization. ImageMagick is distributed under a Apache-style license, which is approved by the Open Source Initiative. ImageMagick is available for free, may be used to support both open and proprietary applications, and may be redistributed without fee, subject to the conditions of the license supplied with the software. It is available for a wide variety of platforms, and can be used in conjunction with APLX under Windows, Linux, and MacOS X (*APLX cannot interface to it under Mac OS 9*). For more information, visit <http://www.imagemagick.org>.

Installation of ImageMagick

To use the Image object, you must have ImageMagick version 6.1 or later installed on your system. If it is not already installed, when you try to create the Image object, APLX will put up a dialog box to report the error. An APL `DOMAIN ERROR` will also be raised.

ImageMagick may be installed already on your system (for example, many Linux distributions include it as standard). If not, the easiest way to install ImageMagick is to download pre-built Binary releases from <http://www.imagemagick.org>, or a mirror site. Versions for Windows, MacOS and Linux are available. We recommend you use the 8-bit versions (8 bits per pixel color). At the time of writing, the files you need to install are:

Windows: ImageMagick-6.1.9-Q8-windows-dll.exe ('dynamic' version)

Linux: ImageMagick-i686-pc-linux-gnu.tar.gz

MacOS X: ImageMagick-powerpc-apple-darwin7.8.0.tar.gz

If you have installed ImageMagick and APLX still reports that it cannot find it, the most likely reason is that the shared library path is not set up correctly. Check first that the ImageMagick utility programs (such as 'display' and 'convert') work correctly outside APLX.

Under Windows, the ImageMagick directory (under 'c:\Program Files') should be in the path. Under Linux, the ImageMagick shared library files should be in /usr/lib.

Supported Graphics Formats

The exact set of supported file formats will depend on the platform, the ImageMagick version, and on whether you have installed extensions called 'delegates'. Each format is identified by a tag string (also used, by default, as the file extension). Some of the most important formats include:

<i>Tag</i>	<i>Description</i>
BMP	Microsoft Windows bitmap
EMF	Microsoft Enhanced Metafile (32-bit)
EPS	Adobe Encapsulated PostScript
GIF	CompuServe Graphics Interchange Format
JPEG	Joint Photographic Experts Group JFIF format
PDF	Portable Document Format
PICT	MacOS PICT file
PNG	Portable Network Graphics
SVG	Scalable Vector Graphics
TIFF	Tagged Image File Format

Not all formats will necessarily be available, and some may be read-only (i.e you can read the format into the image, but not write it out in that format). Note that APLX itself does not contain any format-conversion code for the Image object; this is done in the ImageMagick package or 'delegates', which you install separately.

You can determine which formats are supported by reading the `formats` property of an Image object.

Loading or creating an image

You can either load an existing image from file, or from an APL pixel array, or from the Clipboard, or you can create it from scratch using the `Draw` method of the Image itself.

`Load` method: This takes a single argument, which is a character vector containing the full filename of an image file, in one of the supported formats, which you want to display or manipulate. If you specify an empty vector for the file name, APLX will display a dialog asking the user to select an image file. You can also use pseudo-filenames for certain built-in images and patterns, such as `'magick:logo'` or `'magick:rose'`. See the ImageMagick documentation for more details. The current image, if any, is replaced (unless you have set the `style` property so that images are appended).

`bitmap` property: This is an integer matrix, with one element per pixel. Each pixel is encoded as 256 τ Blue Green Red. If you write to this property, any current image is replaced with the pixels you specify (unless you have set the `style` property so that new images are appended). You can create the pixel array directly in APL, or read it from the `bitmap` property of the System object (read from Clipboard), of a Chart object, or of a Picture object.

`Paste` method: This method takes no arguments. It causes any picture on the Clipboard to be loaded into the Image.

In order to create the image from scratch using the `Draw` method, you need first to set the image size using the `imagesize` property. You then use the `Draw` method in the normal way.

Saving or reading back an image

Similarly, you can save the image to a file, read it back into an APL pixel array, or transfer it to the Clipboard:

`Save` method: This method again takes a character vector argument, which is the file name under which you want to save the image. You can specify the format in which you want to save it by setting the `format` property before calling this method, or alternatively allow the format to be determined automatically from the file extension. If you specify an empty vector for the filename, APLX displays a dialog allowing the user to specify the file name and type.

In this example, an invisible Image object is used to load an image file in JPEG format, and save it out as a PDF (Adobe Acrobat) file:

```
Img←'⍀' ⍵NEW 'Image'
Img.Load 'c:\temp\sds.jpg'
Img.Save 'c:\temp\sds.pdf'
```

`bitmap` property: As well as writing the image out to file, you can also read it back at any time from the `bitmap` property, as an integer array of pixel values. (The size of the returned array will be the same as the internal image size, as returned by the `imagesize` property). You can then manipulate the pixel array further in APL, or write it to the Clipboard using the System object, or display it on a

Picture object. You can also display it on a window (or other control), or print it, using the `Draw` method `GetBitmap` keyword.

`Copy`: This method takes no arguments. It causes the currently-active image to be copied to the Clipboard as a bitmap.

Main properties of the Image object

`imagesize` or `bitmapsizesize`: This is a two-element numeric vector, which contains the height and width of the image, in the current `scale` units. (We recommend that you always use `scale 5`, pixels, for Image objects). In most cases, the initial size of the image is set automatically when you `Load` the image from file, or write an array to the `bitmap` property. If you are creating a new image from scratch using the `Draw` method, you should first specify the size of the image by writing to the `imagesize` property; this will create a new image of the size specified, using the current background color. If you write a new value to the `imagesize` property of an existing image, it will be re-scaled to the new size.

`file`: A character vector, which contains the name of the image file from which the image was loaded (if any). Writing to this property causes the image to be loaded from the named file. Writing an empty vector will bring up a dialog allowing the user to chose the name.

`format`: A character string, which is the tag associated with the file format to be used when the Image is saved to file, for example 'BMP' or 'GIF' or 'PDF'. (It should be one of the values returned by the `formats` property). When you load a graphics file, this property will initially be set to the original format. If you change it, when the file is saved the new format will be used. (You can also set the format implicitly by the file extension).

`formats`: *Read-only* Returns a nested vector of character strings, which are the tags for the formats supported by the underlying ImageMagick package.

`quality`: Some image formats allow you to specify an image-quality parameter, where higher values give a better-quality image but require a larger file size. This integer-scalar property allows you to set or retrieve the quality parameter. The exact meaning depends on the image format. See the ImageMagick documentation for more details.

`colorback` or `colourback`: Sets the background color for the image. This is used to fill the new area when a creating a new image by setting the `imagesize` property, and for filling any new regions which are created when you transform the image, for example in a rotate or shear operation. You can specify it either as a three-element vector of Red Green Blue values (each in the range 0 to 255), or as a single integer of 256 + Blue Green Red. When you read the property, APLX returns the single-integer form.

Handling multiple images within a single Image object

A single Image object may contain multiple images, although only some file formats support this feature. The following properties apply to multiple images:

`imagecount`: *Read-only* An integer scalar, containing the number of images in the object.

`imageindex`: An integer scalar, which specifies the currently active image, in index origin 1. This is therefore a number in the range 1 to `imagecount`. The currently-active image is the one on which any transformations or drawing takes place, and is also the one which will be displayed by the parent Picture object, if appropriate. (You can thus display an animated image by using a timer to advance the `imageindex` value at regular intervals.)

`style`: This is an integer scalar. The default value of 0 means that, when you create a new image using the `Load` or `Paste` method, or by writing to the `bitmap` property, the currently-active image is replaced. If the `style` property is 1, the new image will be appended to the end of the image list, and will become the new active image. This allows you to create a sequence of images (for example, an animated GIF), directly from APL.

The Transform method

ImageMagick includes a very wide range of functions for transforming an image. For example, you can shear, scale, crop, rotate or skew the image, sharpen edges, adjust colors and intensity, and filter out noise. You can also apply special effects to the image, such as making it look like a charcoal drawing or an oil painting. The `Transform` method of the APLX Image object allows you to use many of these functions directly.

See the documentation on the `Transform` method for more information.

Other Methods

`Draw`: This is fully supported for Image objects, allowing you to add graphics elements and text to an existing or new image. See the documentation on the `Draw` method for details.

`Setopacity`: ImageMagick supports images which are transparent, which means that when you place one image on top of another using the `Overlay` method, the background image will still be partially visible through the foreground image. Some file formats, such as PNG, allow specific parts of the image to be transparent. The `Setopacity` method allows you to specify that the whole image is transparent. It takes a single argument, which is a number between 0.0 (completely transparent) to 1.0 (fully opaque, i.e. the background is not visible at all).

`Overlay`: This method allows you to overlay one image (the foreground) on top of another (the background). If the foreground image is transparent, the background image will partially show through it. Normally you first load the background image into the Image object, and then use the `Overlay` method to load the foreground image (from file) on top of it, at a specified position. You can also overlay an image from another Image object. See the documentation on the `Overlay` method for details.

Calling ImageMagick directly

The ImageMagick package includes many other features which are not directly supported by the Image object. You can access these features either using `WNA`, or by writing a small Auxiliary Processor. In order to do this, you need the handle to the underlying ImageMagick 'Magick Wand'; this can be read from the `handle` property.

Interaction between the Image object and the Picture object

The Image object itself is always invisible. However, if you create it as the child of a Picture object, the Picture object will display it. Whenever the Image object is changed, the Picture object will be re-drawn to reflect the changes.

If you are making a series of changes to the Image object, APLX attempts to buffer these up to avoid multiple redraws. However, for best performance when making several changes to an Image, and to avoid flickering effects, you can temporarily set the `update` property of the Picture object to 0 before changing the Image. This suppresses automatic redrawing. When the changes are complete, you should set the `update` property back to 1, which will cause the Image to be redrawn with all the changes you have made.

Example

```

▽DEMO_Image;DEMO;X
[1]  ⍎ Sample function demonstrating use of the Image object
[2]  ⍎ You must have ImageMagick installed on your system to run this demo
[3]  ⍎
[4]  ⍎ Create window with a Picture, which we'll use to display Image object
[5]  DEMO←'□' □NEW 'Window' ♦ DEMO.scale←5 ♦ DEMO.size←376 280
[6]  DEMO.title←'Image'
[7]  DEMO.Pic.New 'Picture' ♦ DEMO.Pic.align←¯1
[8]  ⍎
[9]  ⍎ Create the Image object, as a child of the Picture.
[10] ⍎ This effectively makes the image visible.
[11] ⍎ Use error trapping in case ImageMagick is not installed.
[12] :Try
[13]   DEMO.Pic.Img.New 'Image'
[14] :CatchAll
[15]   'You must have ImageMagick installed to run this demo.'
[16]   'See http://www.imagemagick.org'
[17]   →0
[18] :EndTry
[19] DEMO.Pic.Img.scale←5
[20] ⍎
[21] ⍎ Load a picture into the Image object. This could come from file,
[22] ⍎ in any format supported by ImageMagick (JPEG, GIF, BMP, SVG etc).
[23] ⍎ Here we use an array of pixels, encoded as 256 τ Blue Green Red
[24] DEMO.Pic.Img.bitmap←SAMPLE_BITMAP
[25] ⍎
[26] ⍎ Show the original image for a few seconds
[27] DEMO.Show
[28] X←□DL 3
[29] ⍎
[30] ⍎ Rescale to twice original size, and flip around vertical axis
[31] ⍎ We'll suppress updates temporarily so we don't get a redraw
[32] ⍎ until both operations are done
[33] DEMO.Pic.update←0
[34] DEMO.Pic.Img.Transform 'magnify'
[35] DEMO.Pic.Img.Transform 'flop'
[36] DEMO.Pic.update←1 ⍎ Now do redraw
[37] X←□DL 3
[38] ⍎
[39] ⍎ Now use one of ImageMagick's image-transformation functions
[40] ⍎ (see workspace 10 HELPTRANSFORM for more examples):

```

```

[41] DEMO.Pic.Img.Transform 'charcoal' 1 2
[42]  #
[43]  # Use the Draw method to add text to the Image object
[44] DEMO.Pic.Img.Draw 'Color' 255
[45] DEMO.Pic.Img.Draw 'Text' 'APLX and ImageMagick' 2 60
[46]  #
[47]  # Wait a few seconds, then allow user to save the image to file,
[48]  # in any of the formats supported by ImageMagick (JPEG, GIF, PDF, BMP etc)
[49]  # Providing an empty file name automatically brings up the dialog
[50] X←DDL 3
[51] DEMO.Pic.Img.Save ''
[52]  #
[53]  # Window will be deleted when function ends and DEMO goes out of scope
      v

```

Properties

bitmap bitmapsizes children class colorback data events file format formats handle imagecount
 imageindex imagesize methods name opened properties quality self style tie

Methods

Close Copy Create Delete Draw Load New Open Overlay Paste Save Send Set Setopacity Transform
 Trigger

Callbacks

onClose onDestroy onOpen onSend

ImageList

Not implemented under MacOS

Description

The ImageList class allows you to attach images to Menus, Tree controls, and the Pages (tab sheets) in a Selector control.

As its name implies, an ImageList object contains a series of images (typically small icons or pictures), numbered from 1 onwards. These images are all of the same size. The ImageList object is not itself visible; instead, you associate an ImageList with another control, and that control displays images from the list as required. For example, you can have a Tree control where different images are used for different types of node, or you can display an icon from the ImageList next to a Menu item.

The ImageList can be created either as a top-level object, or as the child of another object. The former case is preferable if it will be used in several different controls. On the other hand, if it will be used in only one control, then it may be more convenient to create it as a child of that control so that it is automatically deleted with the control.

You can specify the images in the ImageList either as file names, or as APL numeric arrays where each element contains the color corresponding to a pixel in the image. The `imagenames` property or `Addimages` method can be used to specify the images. You can arrange for a portion of the image to be transparent.

A single bitmap can optionally contribute multiple images to the ImageList, laid side by side. To do this, you need first to set the `imagesize` property to be the size of the image. Any bitmap of a width `N` times the width specified in `imagesize` will then contribute `N` images to the list.

Once you have created an ImageList, you associate it with the control where it will be used by writing its name to the `imagelist` property of a Tree, Selector or pop-up Menu control. (For a Tree control there is also a second ImageList contained in the `imagelistuser` property). To use an ImageList for the menu items in a window's menu bar, write its name to the `menuimagelist` property of the Form, Dialog, Document or Window object. A given ImageList can be used by several other objects.

Finally, you need to tell the control associated with the ImageList to display an image from the list. For a Menu item or Page control, you do this by specifying the `imageindex` property, as an integer index (in origin 1) into the images in the ImageList. For a Tree control, there are various options; you can specify a fixed image for particular nodes, or the image shown can depend on the state of the node. See the `list` property for more information.

Cross-platform use: Although this class is implemented under both Windows and Linux, there are some differences in the supported properties and methods. It is not implemented at all under MacOS.

ImageList properties

The main ImageList properties are:

`style`: The default style is 0, meaning that the images are not automatically transparent. If you set the style to 1, pixels corresponding to the `maskcolor` property are treated as transparent.

`maskcolor`: Allows you to specify the color which will be treated as transparent if `style` is 1. The color can be a single integer 256 + Blue Green Red, or a vector of Red Green Blue values separately (in range 0 to 255).

`imagenames`: Allows you to specify a vector of file names, one for each image in the list. Alternatively, you can specify APL arrays which encode the pictures directly as pixel color values. You can also explicitly set the transparency mask for an image.

`imagesize`: Specifies the height and width of the images, in the current `scale` of the ImageList.

`overlays`: Allows you to specify the index positions (in origin 1) of up to four images in the ImageList which will be used to overlay other images. These overlays can then be used in a Tree object. *Note: This property is not available under Linux.*

`imagecount`: A read-only property, the number of images in the list.

ImageList methods

The main ImageList method is:

`Addimages`: Allows you to add images to the list.

Example 1: Using an ImageList in a window's menu

```

    ▽DEMO_Menu_Images; img; DEMO; MyImageList
[1]  ⍝ Use of an ImageList and a Menu
[2]  ⍝
[3]  DEMO←'⎕' ⎕NEW 'Window' ⋄ DEMO.title←'Menu & ImageList Example'
[4]  ⍝
[5]  ⍝ Create an image list containing icons and add it to the window
[6]  MyImageList←'⎕' ⎕NEW 'ImageList'
[7]  MyImageList.imagesize←16 16
[8]  MyImageList.maskcolour←0 ⍝ Black areas transparent
[9]  img←16 16⍲0 ⍝ Create a 16×16 pixel colour matrix
[10] img[4+⍲8;4+⍲8]←256⍲0 255 0 ⍝ Make middle 8x8 green
[11] MyImageList.AddImages img ⍝ Image 1: green square
[12] img[4+⍲8;4+⍲8]←256⍲255 0 0 ⍝ Make middle 8x8 blue
[13] MyImageList.AddImages img ⍝ Image 2: blue square
[14] MyImageList.maskcolour←(256⍲255 255 255) ⍝ White areas transparent
[15] MyImageList.AddImages(FULL_IMAGE_PATH 'abc.bmp') ⍝ Images 3,4,5 from file

```

```

[16] DEMO.menuimagemelist←MyImageList.name
[17] ⍺
[18] ⍺ Set up the File menu
[19] ⍺ The 'imageindex' property of a window's menu items select the image
[20] ⍺ from the 'menuimagemelist' of the window
[21] ⍺
[22] DEMO.FileMenu.New 'Menu' ⋄ DEMO.FileMenu.caption←'&File'
[23] DEMO.FileMenu.imageindex←1
[24] DEMO.FileMenu.Open.New 'Menu' ⋄ DEMO.FileMenu.Open.caption←'&Open'
[25] DEMO.FileMenu.Open.imageindex←3
[26] DEMO.FileMenu.Close.New 'Menu' ⋄ DEMO.FileMenu.Close.caption←'&Close'
[27] DEMO.FileMenu.Close.imageindex←4
[28] DEMO.FileMenu.Exit.New 'Menu' ⋄ DEMO.FileMenu.Exit.caption←'&Exit'
[29] DEMO.FileMenu.Exit.shortcut←(81 2)
[30] DEMO.FileMenu.Exit.imageindex←5
[31] ⍺
[32] ⍺ Set up the Edit menu
[33] DEMO.EditMenu.New 'Menu' ⋄ DEMO.EditMenu.caption←'&Edit'
[34] DEMO.EditMenu.imageindex←2
[35] DEMO.EditMenu.Undo.New 'Menu' ⋄ DEMO.EditMenu.Undo.caption←'&Undo'
[36] DEMO.EditMenu.Undo.enabled←0 ⋄ DEMO.EditMenu.Undo.shortcut←(92 2)
[37] DEMO.EditMenu.Sep1.New 'Menu' ⋄ DEMO.EditMenu.Sep1.separator←1
[38] DEMO.EditMenu.Cut.New 'Menu' ⋄ DEMO.EditMenu.Cut.caption←'&Cut'
[39] DEMO.EditMenu.Cut.shortcut←('C' 2)
[40] DEMO.EditMenu.Cut.imageindex←1
[41] DEMO.EditMenu.Copy.New 'Menu' ⋄ DEMO.EditMenu.Copy.caption←'&Copy'
[42] DEMO.EditMenu.Copy.shortcut←('X' 2)
[43] DEMO.EditMenu.Copy.imageindex←2
[44] DEMO.EditMenu.Paste.New 'Menu' ⋄ DEMO.EditMenu.Paste.caption←'&Paste'
[45] DEMO.EditMenu.Paste.shortcut←('V' 2)
[46] ⍺
[47] ⍺ Wait for the user to close the window
[48] 0 0ρ␣WE DEMO
      ▾

```

Example 2: Using an ImageList in a pop-up menu

```

      ▾DEMO_Popup_Images; img; MyPopUp; il
[1] ⍺ Demonstrate pop-up menu in conjunction with the ImageList System Class
[2] ⍺
[3] ⍺ Create the pop-up menu itself
[4] MyPopUp←'␣' ␣NEW 'Menu'
[5] ⍺
[6] ⍺ Create an imagelist object to provide images for the menu items
[7] ⍺ and add it to the popup menu
[8] il←'␣' ␣NEW 'ImageList'
[9] il.imagesize←16 16
[10] ⍺
[11] ⍺ Create image 1: Circle
[12] img←16 16ρ256␣255 255 255 ⍺ Create a 16×16 pixel colour matrix
[13] img[1,16;]←256␣255 0 0 ⋄ img[;1,16]←256␣255 0 0 ⍺ Surround by blue square
[14] img[6,9;7,8]←0 ⋄ img[7,8;6,9]←0
[15] il.AddImages img
[16] ⍺
[17] ⍺ Create image 2: Triangle
[18] img←16 16ρ256␣255 255 255 ⍺ Create a 16×16 pixel colour matrix
[19] img[1,16;]←256␣255 0 0 ⋄ img[;1,16]←256␣255 0 0 ⍺ Surround by blue square
[20] img[6,7,8,9;6]←0 ⋄ img[9;6,7,8,9]←0 ⋄ img[7;7]←0 ⋄ img[8;8]←0
[21] il.AddImages img
[22] ⍺

```

```

[23] A Create image 3: Line
[24] img←16 16p256⍲255 255 255 A Create a 16×16 pixel colour matrix
[25] img[1,16;]←256⍲255 0 0 ⋄ img[;1,16]←256⍲255 0 0 A Surround by blue square
[26] img[7;6,7,8,9]←0
[27] il.AddImages img
[28] A
[29] A Create image 4: Rectangle
[30] img←16 16p256⍲255 255 255 A Create a 16×16 pixel colour matrix
[31] img[1,16;]←256⍲255 0 0 ⋄ img[;1,16]←256⍲255 0 0 A Surround by blue square
[32] img[6;6,7,8,9]←0 ⋄ img[9;6,7,8,9]←0 ⋄ img[6,7,8,9;5]←0
[33] img[6,7,8,9;10]←0
[34] il.AddImages img
[35] A
[36] A Assign the image list to the popup menu
[37] MyPopUp.imageList←il.name
[38] A
[39] A Create the pop-up menu items
[40] A The 'imageindex' property of a popup menu's items select the image
[41] A from the 'imageList' of the popup menu
[42] MyPopUp.Rect.New 'Menu' ⋄ MyPopUp.Rect.caption←'Rectangle'
[43] MyPopUp.Rect.imageindex←4
[44] MyPopUp.Circle.New 'Menu' ⋄ MyPopUp.Circle.caption←'Circle'
[45] MyPopUp.Circle.imageindex←1
[46] MyPopUp.Triangle.New 'Menu' ⋄ MyPopUp.Triangle.caption←'Triangle'
[47] MyPopUp.Triangle.imageindex←2
[48] MyPopUp.Line.New 'Menu' ⋄ MyPopUp.Line.caption←'Line'
[49] MyPopUp.Line.imageindex←3
[50] A
[51] A In a real example, you would define onClick callbacks here
[52] A
[53] A Pop up the menu at the mouse position
[54] MyPopUp.Popup
v

```

Example 3: Using an ImageList with a Selector and Pages

```

vDEMO_Page_Images;DEMO
[1] A Demonstrating use of the Selector (tab sheet) System Class
[2] A in conjunction with the ImageList class
[3] A
[4] DEMO←'□' □NEW 'Dialog' ⋄ DEMO.scale←1
[5] DEMO.title←'Selector/Page & ImageList Example'
[6] A Create selector to hold the individual pages, fill the client area
[7] DEMO.Sel.New 'Selector' ⋄ DEMO.Sel.align←_1
[8] A
[9] A Create image list with icons for the pages and add to selector.
[10] A (Image list does not have to be child of the selector, but it
[11] A is convenient because it will be deleted with its parent)
[12] DEMO.Sel.il.New 'ImageList'
[13] DEMO.Sel.il.imagesize←16 16
[14] DEMO.Sel.il.maskcolour←(256⍲255 255 255) A White areas transparent
[15] DEMO.Sel.il.AddImages(FULL_IMAGE_PATH 'abc.bmp')
[16] DEMO.Sel.imageList←DEMO.Sel.il.name
[17] A
[18] A Now create three pages each with some radio buttons
[19] DEMO.Sel.Page1.New 'Page' ⋄ DEMO.Sel.Page1.caption←'Appetizer'
[20] DEMO.Sel.Page1.imageindex←1 A Use image 1 in selector's image list
[21] DEMO.Sel.Page1.R1.New 'Radio'
[22] DEMO.Sel.Page1.R1.caption←'Smoked Salmon'
[23] DEMO.Sel.Page1.R1.where←2 1
[24] DEMO.Sel.Page1.R1.value←1

```

```

[25] DEMO.Sel.Page1.R2.New 'Radio'
[26] DEMO.Sel.Page1.R2.caption←'Fresh asparagus'
[27] DEMO.Sel.Page1.R2.where←4 1
[28] DEMO.Sel.Page1.R3.New 'Radio'
[29] DEMO.Sel.Page1.R3.caption←'Clam chowder'
[30] DEMO.Sel.Page1.R3.where←6 1
[31] ␣
[32] DEMO.Sel.Page2.New 'Page' ⋄ DEMO.Sel.Page2.caption←'Main course'
[33] DEMO.Sel.Page2.imageindex←2
[34] DEMO.Sel.Page2.R1.New 'Radio'
[35] DEMO.Sel.Page2.R1.caption←'Roast duck'
[36] DEMO.Sel.Page2.R1.where←2 1
[37] DEMO.Sel.Page2.R1.value←1
[38] DEMO.Sel.Page2.R2.New 'Radio'
[39] DEMO.Sel.Page2.R2.caption←'Grilled steak'
[40] DEMO.Sel.Page2.R2.where←4 1
[41] DEMO.Sel.Page2.R3.New 'Radio'
[42] DEMO.Sel.Page2.R3.caption←'Fresh sea-bass'
[43] DEMO.Sel.Page2.R3.where←6 1
[44] ␣
[45] DEMO.Sel.Page3.New 'Page'
[46] DEMO.Sel.Page3.caption←'Dessert'
[47] DEMO.Sel.Page3.imageindex←0 ␣ Use no image for this page
[48] DEMO.Sel.Page3.R1.New 'Radio'
[49] DEMO.Sel.Page3.R1.caption←'Fresh stawberries'
[50] DEMO.Sel.Page3.R1.where←2 1
[51] DEMO.Sel.Page3.R1.value←1
[52] DEMO.Sel.Page3.R2.New 'Radio'
[53] DEMO.Sel.Page3.R2.caption←'Creme caramel'
[54] DEMO.Sel.Page3.R2.where←4 1
[55] DEMO.Sel.Page3.R3.New 'Radio'
[56] DEMO.Sel.Page3.R3.caption←'Baked Alaska'
[57] DEMO.Sel.Page3.R3.where←6 1
[58] ␣
[59] ␣ Wait for the user to close the window
[60] 0 0ρ□WE DEMO
      ▽

```

Properties

children class data events handle imagealloc imagecount imagenames imagesize maskcolor methods name opened overlays properties self style tie

Methods

Addimages Close Create Delete New Open Send Set Trigger

Callbacks

onClose onDestroy onOpen onSend

Label

Alternative name: Static

Description

The Label (or Static) class implements simple text labels.

Normally you need to set only the `caption` property, which determines the text displayed, and the `where` property, to determine where it is placed on the window. You can also set the `font`, `color` and `enabled` properties for special effects (if `enabled` is 0, the text is grayed out).

Under Windows and Linux only. the `style` property for a static text object sets the text alignment:

```
0 = Left justified
1 = Centered
2 = Right justified
```

plus optional bits:

```
4 = Inhibit prefix (interpretation of '&' as accelerator key (Windows, Linux only)
8 = Inhibit word-wrap if the text is too wide for the label
32 = Inhibit auto-sizing of the label to ensure text fits
128 = Text is transparent (Windows, Linux only)
```

Event handling is not usually relevant to Label objects, but sometimes it is useful to define an `onMouseDown` callback. In this case, you should usually provide some visual indication to the user that the text is capable of responding to mouse events, for example by displaying it in a special color or underlining it using the `font` property. This can be used for 'hypertext' type applications where the user clicks on a word for more information about it.

Note: Another way of displaying text on a window or control is to use the `Draw` method. This provides more flexibility in specifying the font colors and transparency, and also allows you to write text at any angle. Under Windows, it is also useful where you want the text to appear over another control; Label objects always display *behind* windowed controls such as Buttons.

Example

```
∇DEMO_Label;DEMO
[1]  ⍠ Sample function demonstrating use of the Label object
[2]  DEMO←'␣' ⓂNEW 'Window' Ⓜ DEMO.scale←1
[3]  DEMO.title←'Label Example'
[4]  ⍠
[5]  DEMO.LABEL1.New 'Label'
[6]  DEMO.LABEL1.caption←'Today's Work Plan'
[7]  DEMO.LABEL1.where←2 1 1 30
[8]  DEMO.LABEL1.font←'Helvetica' 1 1 ⍠ Bold text
[9]  ⍠
```

```

[10] DEMO.LABEL2.New 'Label'
[11] DEMO.LABEL2.caption<'Tidy up the office!'
[12] DEMO.LABEL2.where<4 1 1 30
[13] DEMO.LABEL2.enabled<0
[14] #
[15] DEMO.LABEL3.New 'Label'
[16] DEMO.LABEL3.caption<'Have lunch'
[17] DEMO.LABEL3.where<6 1 1 30
[18] DEMO.LABEL3.font<'Helvetica' 1 0
[19] DEMO.LABEL3.color<255
[20] #
[21] # Wait for the user to close the window
[22] 0 0ρWE DEMO
▽

```

Properties

align anchors aquaadjust autodraw caption children class color data dragsource droptarget enabled events extent font maxsize methods minsize name opened pointer properties scale self size sourceformats style targetformats tie tooltip units visible where

Methods

Click Clienttoscreen Close Create Delete Hide New Open Paint Resize Screentoclient Send Set Show Trigger

Callbacks

onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave onDragOver onDragStart onHide onMouseDown onMouseMove onMouseUp onOpen onSend onShow

Line

Description

The Line object draws a single line, and is normally used for display purposes only. The `where` or `size` property is used to define the enclosing rectangle, with a special meaning. The first two parameters (top and left of the 'rectangle') are the starting position of the line. The second two (height and width of the 'rectangle') are the offset to the end of the line, and may be negative. For example, if `where` is `20 20 10 0`, the line starts at `20 20` and is vertically downwards; if `where` is `20 20 -10 0`, it is vertically upwards.

You can set the drawing (foreground) color using the `color` property, and the `pensize` property changes the thickness of the lines.

Note: Remember that the default scale is in character units. See the `scale` property for details.

See also the Draw method which allows you to draw geometric shapes on your windows and controls.

Example

```

    ▽DEMO_Line;DEMO
[1]  a Sample function demonstrating use of the Line object
[2]  DEMO←'□' □NEW 'Window' ◇ DEMO.scale←1
[3]  DEMO.title←'Line Example'
[4]  a
[5]  a Thin horizontal line in red
[6]  DEMO.LN1.New 'Line'
[7]  DEMO.LN1.where←4 0 0 10
[8]  DEMO.LN1.color←255
[9]  a
[10] a Thicker diagonal line in blue
[11] DEMO.LN2.New 'Line'
[12] DEMO.LN2.where←4 10 8 10
[13] DEMO.LN2.pen←4
[14] DEMO.LN2.color←0 0 255
[15] a
[16] a Wait for the user to close the window
[17] 0 0□WE DEMO
    ▽

```

Properties

align anchors aquaadjust autodraw children class color data dragsource droptarget enabled events extent filled maxsize methods minsize name opened pointer properties scale self size sourceformats targetformats tie units visible where

Methods

Click Clienttoscreen Close Create Delete Hide New Open Paint Resize Screentoclient Send Set Show Trigger

Callbacks

onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave onDragOver onDragStart onHide onMouseDown onMouseMove onMouseUp onOpen onSend onShow

List

Description

The List class implements list boxes, allowing the user to make a selection from a list.

The `style` property defines the type of list box, as the sum of the following flags:

```

1 = Allow multi select
2 = Allow display of a non-integral number of lines
16 = Show the vertical scroll bar
64 = Show the horizontal scroll bar (Mac only)
128 = Hide scroll bars when they are not needed (Mac only)

```

The `list` property determines the items in the list box. You can set it either as a character matrix or as a character vector with embedded carriage returns. (If you read it back, it will be returned as a character matrix).

The `value` property is an integer scalar or vector containing the index into the list of the selected item, in index origin 1. It is an empty vector if there is no selection. If the `style` allows multi-select, it may be a vector of more than one element.

The `firstvisible` property allows you to read or set the first visible item in the list (i.e. where the list is scrolled to).

If you need to detect when the user changes the selection, add an `onChange` callback.

Example

```

▽DEMO_List;DEMO
[1]  ⍝ Sample function demonstrating use of the List object
[2]  DEMO←' ' ⍵NEW 'Dialog' ⋄ DEMO.scale←1
[3]  DEMO.title←'List Example'
[4]  DEMO.myList.New 'List'
[5]  DEMO.myList.where←2 1
[6]  DEMO.myList.list←⍵M
[7]  DEMO.myList.value←3
[8]  ⍝
[9]  ⍝ Wait for the user to close the window
[10] 0 0ρ⍵WE DEMO
▽

```

Properties

align anchors aquaadjust autodraw border caption children class color data doublebuffered dragsource droptarget enabled events extent firstvisible font handle list maxsize methods minsize name opened order pointer properties scale self size sourceformats style tabstop targetformats tie units value visible where winptr

Methods

Click Clienttoscreen Close Create Delete Draw Focus Hide New Open Paint Resize Screentoclient
Send Set Show Trigger

Callbacks

onChange onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter
onDragLeave onDragOver onDragStart onFocus onHide onKeyDown onKeyPress onKeyUp
onMouseDown onMouseMove onMouseUp onOpen onSend onShow onUnFocus

Menu

Description

The Menu class implements menu bars, sub-menus, and individual menu items. It also implements pop-up menus. The behavior depends on the *parent* of the Menu object:

- If the Menu object is created as the child of a window object, it represents an item in the menubar of the window, for example the 'File' menu of a typical application window.
- If the Menu object is defined as top-level object, it is a pop-up menu which can be displayed using the `Popup` method.
- If the Menu object is the child of another Menu object, it is an item in the parent menu or sub-menu. Sub-menus can be multiple levels deep if required.

Principal Menu properties

The main properties used when working with Menu objects are:

`caption`: A character vector containing the menu title string. An ampersand character '&' in the string will not be displayed, but under Windows will cause the next character to be the keyboard-selection character for the menu. Under MacOS, the ampersand will have no effect but again will not be displayed, so you can use the same string for cross-platform applications. To display an ampersand in the menu title, put two ampersands in the character vector.

`enabled`: 1 if menu is enabled, 0 to disable it (default: 1)

`visible`: 1 if menu is visible, 0 to hide it (default: 1)

`value`: 1 to place a check mark by the menu item, else 0 (default 0)

`separator`: 1 if the item is just a separator line, else 0

`shortcut`: The shortcut key (if any) as a two-element vector. The first element is the letter corresponding to the key (for example, 'A'), or alternatively an integer which is the virtual key code (for alphabetic keys, this is just the ASCII code, for example 65 for 'A'). Under Windows and Linux, the second element is the modifier code: 0 (plain), 1 (shift), 2 (ctrl), 4 (alt) and sums thereof. *Note: Some of these may not be useful in practice because the operating-system intervenes before APLX sees the keystroke; in Windows, for example, this happens with the Alt key.* The second element can be omitted and defaults to 2 (ctrl). Under MacOS, the second element is ignored since the Command key is always used for menu shortcuts. If the `shortcut` property is an empty vector, there is no shortcut key.

Other Menu properties

Other less commonly-used properties include:

`order`: Position of menu item in parent menu, in index origin 1. Set by default to the order of creation, but you can change it under program control (under Windows only). Set a fractional value to insert a menu item between two existing items. The menu items will be renumbered from 1 to the total number of items. Writing to this property has no effect under MacOS.

`group`: Menu items can automatically work like radio buttons, i.e. if one of the group is selected, all others are automatically deselected. Normally, this behavior is switched off, with the `group` property set to 0. Setting the `group` property to an integer in the range 1 to 255 makes the the item a member of that group, and enables the radio-button behavior.

`style`: If the `style` property is 1, the menu item is automatically selected/deselected when the user clicks it (the `value` property will reflect the state). This may sometimes mean that there is no need to define an `onClick` handler for the menu, because you can read back its `value` property when you need it.

Pop-up Menu Methods

When you want to display the pop-up menu (for example, in response to a right-mouse click), you call the `popup` method. If you call this without parameters, the menu pops up at the mouse position. Alternatively you can supply the Top and Left coordinates as a two-element vector in the parent (System object's) scaling units.

Menu callbacks

You typically respond to menu selections by defining the `onClick` callback of the menu item, to carry out the appropriate action when the user selects a menu. An alternative way of doing this is to define an `onMenu` callback for the window containing the menu bar; this allows a single callback function to respond to a whole set of menu items. You can combine the two approaches if it is more convenient to do so.

Placing images next to menu items

Under Windows and Linux, you can optionally place images next to menu items. To do this, you need to create an `ImageList` object which contains all the images which appear in the menubar (or pop-up menu) and any sub-menu items. You then associate the `ImageList` with the menu hierarchy as follows:

- For the menus in a window's menubar, write the name of the `ImageList` to the window's `menuimagelist` property.
- For a pop-up menu, write the name of the `ImageList` to the pop-up menu's `imagelist` property.

Finally, you set the `imageindex` property of each menu item where you want to display an image. This selects one of the images from the `ImageList`. (See the description of the `ImageList` object for an example.)

Example 1: Window menu bar

```

    ▽DEMO_Menu;DEMO
[1]  A Sample function demonstrating use of the Menu object
[2]  DEMO←'□' □NEW 'Window' ◇ DEMO.title←'Menu Example'
[3]  A
[4]  A Set up the File menu
[5]  DEMO.File.New 'Menu' ◇ DEMO.File.caption←'&File'
[6]  DEMO.File.Open.New 'Menu' ◇ DEMO.File.Open.caption←'&Open'
[7]  DEMO.File.Close.New 'Menu' ◇ DEMO.File.Close.caption←'&Close'
[8]  DEMO.File.Exit.New 'Menu' ◇ DEMO.File.Exit.caption←'&Exit'
[9]  DEMO.File.Exit.shortcut←81 2
[10] A
[11] A Set up the Edit menu
[12] DEMO.Edit.New 'Menu' ◇ DEMO.Edit.caption←'&Edit'
[13] DEMO.Edit.Undo.New 'Menu' ◇ DEMO.Edit.Undo.caption←'&Undo'
[14] DEMO.Edit.Undo.shortcut←92 2
[15] DEMO.Edit.Undo.enabled←0
[16] DEMO.Edit.Sep1.New 'Menu' ◇ DEMO.Edit.Sep1.separator←1
[17] DEMO.Edit.Cut.New 'Menu' ◇ DEMO.Edit.Cut.caption←'&Cut'
[18] DEMO.Edit.Cut.shortcut←'C' 2
[19] DEMO.Edit.Copy.New 'Menu' ◇ DEMO.Edit.Copy.caption←'&Copy'
[20] DEMO.Edit.Copy.shortcut←'X' 2
[21] DEMO.Edit.Paste.New 'Menu' ◇ DEMO.Edit.Paste.caption←'&Paste'
[22] DEMO.Edit.Paste.shortcut←'V' 2
[23] A
[24] A Set up callbacks. In a real example, these would be functions.
[25] A We just print the name of the menu which was clicked
[26] DEMO.File.Open.onClick←'□←□WSELF,' selected''
[27] DEMO.File.Close.onClick←'□←□WSELF,' selected''
[28] DEMO.File.Exit.onClick←'□←□WSELF,' selected''
[29] DEMO.Edit.Undo.onClick←'□←□WSELF,' selected''
[30] DEMO.Edit.Cut.onClick←'□←□WSELF,' selected''
[31] DEMO.Edit.Copy.onClick←'□←□WSELF,' selected''
[32] DEMO.Edit.Paste.onClick←'□←□WSELF,' selected''
[33] A
[34] A Wait for the user to close the window
[35] 0 0ρ□WE DEMO
    ▽

```

Example 2: Pop-up menu

```

    ▽DEMO_Popup;MyPopUp
[1]  A Sample function demonstrating pop-up menu
[2]  A Create the pop-up menu itself
[3]  MyPopUp←'□' □NEW 'Menu'
[4]  A
[5]  A Create the pop-up menu items
[6]  MyPopUp.Rect.New 'Menu' ◇ MyPopUp.Rect.caption←'Rectangle'
[7]  MyPopUp.Circle.New 'Menu' ◇ MyPopUp.Circle.caption←'Circle'
[8]  MyPopUp.Triangle.New 'Menu' ◇ MyPopUp.Triangle.caption←'Triangle'
[9]  MyPopUp.Line.New 'Menu' ◇ MyPopUp.Line.caption←'Line'
[10] A
[11] A In a real example, you would define onClick callbacks here
[12] A
[13] A Pop up the menu at the mouse position
[14] MyPopUp.Popup
    ▽

```

Properties

caption children class data enabled events group imageindex imagelist menuimagelist methods name opened order properties self separator shortcut style tie value visible

Methods

Click Clienttoscreen Close Create Delete New Open Popup Send Set Trigger

Callbacks

onClick onClose onDestroy onOpen onPopup onSend

See also onMenu

Movie

Alternative name: Media

Functionality restricted under Linux

Description

The Movie object class is used to play movies, audio clips, and other multi-media files from disk. Movies can be of two kinds, depending on the `style` property (you must set this at the time you create the Movie object, before setting the `file` property). If the Movie has a controller (which is the default, `style = 0`), it is displayed with a VCR-style control along the bottom edge, which lets the user play, stop, change the position in the movie. In this case, your program has no control over the playing of the movie, and no events are created. Alternatively, if the `style` is 1, the movie is played under control of your program using the `Play`, `Stop` and `Rewind` methods, and you can get an `onMovieEnd` callback when it completes. A `style` of 2 is similar, except that the movie once started plays continuously until the `Stop` method is called. No `onMovieEnd` callback is invoked.

In each case, you set the `file` property to determine the movie (or other type of file) which is to be played. If this is set to an empty vector, a standard file selection dialog is displayed to allow the user to choose the movie. Selecting the movie also re-sizes the object to the original size of the movie, though you can change this later if you wish (movies display best if the size is an integral multiple of the original size, but can be stretched or compressed if you wish). The `playing` property is useful for finding out the current movie state.

Cross-platform use: This class is fully implemented under Windows and MacOS, but under Linux only a sub-set of the functionality is available. The Linux implementation can play animated GIF files only (these are commonly used to display animated images in web pages). There is no support for sound. In addition, under Linux there is no movie-controller; style 0 is not implemented.

Example

```

▽DEMO_Movie;DEMO
[1]  # Sample function demonstrating use of the Movie object
[2]  DEMO←'□' □NEW 'Window' ♦ DEMO.title←'Movie Example'
[3]  DEMO.myMovie.New 'Movie' ♦ DEMO.myMovie.where←0 0 ♦ DEMO.myMovie.style←0
[4]  # Set the file property to empty vector to put up selection dialog
[5]  DEMO.myMovie.file←''
[6]  #
[7]  # Wait for the user to close the window
[8]  0 0□WE DEMO
▽

```

Properties

align anchors aquaadjust autodraw caption children class color data doublebuffered dragsource
 droptarget enabled events extent file handle maxsize methods minsize movieref name opened order

playing pointer position properties range scale self size sourceformats style tabstop targetformats tie
units visible volume where winptr

Methods

Click Clienttoscreen Close Create Delete Draw Focus Hide New Open Paint Play Poster Preview
Resize Rewind Screentoclient Send Set Show Stop Trigger

Callbacks

onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave
onDragOver onDragStart onFocus onHide onKeyDown onKeyPress onKeyUp onMouseDown
onMouseMove onMouseUp onOpen onSend onShow onStop onUnFocus

MsgBox

Description

The `MsgBox` class implements the pre-defined message-box dialog. This is a top-level object (i.e. it should not be opened as the child of a window).

To use the dialog, you must first create it as a top-level object using the `New` method; it will remain hidden at this stage. You can then set up the dialog using the following properties:

`caption`: The dialog title as a character vector.

`text`: The text to appear in the message-box, as a character vector.

`style`: The buttons to appear in the message-box, as an integer scalar. Valid values are:

```
0 = OK only
1 = OK and Cancel
2 = Retry and Cancel
3 = Yes and No
4 = Yes, No and Cancel
5 = Abort, Retry and Ignore
```

`icon`: Icon as an integer scalar (default 0):

```
0 = Stop
1 = Alert
2 = Note
3 = Question
-1 = None
```

`default`: The button to use as the default (1 to 4 - see button numbers below)

Displaying the dialog

When you are ready to show the dialog, call the `Show` method. This displays the modal dialog, and waits for the user to press one of the buttons. The `Show` method returns the button which the user pressed to end the dialog:

```
1 = OK
2 = Cancel
3 = Abort
4 = Retry
5 = Ignore
6 = Yes.
7 = No
-1 = The dialog ended without a button being clicked
```

Finally, you will normally call the `Delete` method to dispose of the dialog. Alternatively, you can keep it for re-use later in your application.

Example

```

▽DEMO_MsgBox;DLG
[1]  ⍎ Sample function demonstrating use of the MsgBox object
[2]  DLG←'□' □NEW 'MsgBox' ♦ DLG.icon←2 ♦ DLG.style←1
[3]  DLG.caption←'Sample Message'
[4]  DLG.text←'Sounds fun. Shall we do it?'
[5]  ⍎ Show the dialog. If user presses Cancel, quit
[6]  →(2=DLG.Show)/0
[7]  ⍎ He pressed OK.
[8]  'OK, let''s do it!'
▽

```

Properties

caption children class data default events icon methods name opened properties self style text tie

Methods

Close Create Delete New Open Send Set Show Trigger

Callbacks

onClose onDestroy onOpen onSend

OLEContainer

Implemented under Windows only

Description

The OLEContainer object allows you to include OLE (Object Linking and Embedding) documents in your APLX windows. For example, you could include a Microsoft Word document as part of a more complex window, merging the Word menus and toolbar with your own. The OLEContainer control handles all of the OLE protocol required for this to work; you merely need to specify the document (or type of document) to be embedded or linked, or alternatively you can allow the user to select one. (You can use the 'Control Browser' item in the APLX 'Tools' menu to find out information about the OLE object types available on your system).

The OLEContainer object automatically merges menus defined by the OLE server application with your own menus on the same window. The `group` property of your main menu items determine how the merging takes place. Top-level menu items whose `group` property is one of 0, 2, and 4 are left unchanged. Top-level menu items whose `group` property is one of 1, 3, and 5 are merged with the OLE server applications menus. In addition, OLE objects which are activated inside your windows add their own toolbars to your window.

Once the document is embedded or linked, the OLEContainer object will have additional properties, methods, and events which belong to the external software but which you can access just as you do with ordinary APLX objects. (To avoid any possible ambiguity, each external property, method, or event name is prefixed with an 'x', but you can omit this provided there is no clash with an APLX built-in name.)

More information on using OLE objects can be found in the separate chapter on OCX/ActiveX Controls and OLE Automation.

OLEContainer properties

The main OLEContainer properties are:

`style`: The style is the sum of a set of flags:

1 = Linked instead of embedded document (must be set before setting the 'file' property)
2 = Allow in-place activation
4 = Show as icon instead of document (must be set before setting 'file' property)

`progid`: A character vector determining the class of object (for example, whether it is a Word document or a Pinnacle Chart). It is of the same form as the third column of the `oledoctype`s property of the System object. If there is a document loaded, reading the `progid` property allows you to find out what the document type is. Writing to the `progid` property creates a new empty document of the appropriate class. If you write an empty vector, a dialog is displayed allowing the user to choose.

`file`: A character vector determining the name of the document contained in the `OLEContainer`. If the `OLE` container is linked to a document, reading the `file` property allows you to find out its name. Writing to the `file` property causes the container to attempt to load the document (or link to it, if the `style` is 1). If you write an empty vector, a dialog is displayed allowing the user to choose a document.

`autoactivate`: Determines how the activation of the document takes place. One of:

0 = Manual (use the `DoVerb` method to activate the document)
 1 = Automatic (APLX activates it automatically) [Default]
 2 = Activate when the control gets focus
 3 = Activate when the user double-clicks in the control.

`sizemode`: Determines how the document fits in the container. One of:

0 = Displays the `OLE` object at its normal size, clipping any parts that don't fit within the container. [Default]
 1 = Displays the `OLE` object at its normal size, centering it within the container.
 2 = Scales or shrinks the view of the `OLE` object to fit within the container, by scaling width and height proportionally.
 3 = Scales or shrinks the view of the `OLE` object to fill the `OLE` container, without regard to preserving the proportions of the `OLE` object.
 4 = Displays the `OLE` object at its normal size and automatically resizes the container to fit the size of the `OLE` object.

`docstate`: Read-only property giving information about the `OLE` document state. One of:

0 = There is no `OLE` object in the container.
 1 = There is an `OLE` object in the container, but its server application isn't currently running.
 2 = The `OLE` object's server is running.
 3 = The `OLE` object is open in a separate window.
 4 = The `OLE` object is activated inplace, but hasn't yet merged its menus or toolbars.
 5 = The `OLE` object is activated inplace and menus and toolbars have been merged

`modified`: Boolean property, saying whether the document has been modified. You can reset this to 0 (false).

`verbs`: Read-only property. It returns a nested array of the application-specific 'verbs' which the server application defines. See the `DoVerb` method for details.

`contents`: When you read the `contents` property, APLX returns a character vector containing the internal representation of the document (or the link) in the container, encoded as a series of bytes and including information about the type of the document. You should never change the contents of this vector, but you can store it in a file and write the same vector back to an `OLEContainer` to cause it to re-display the same document.

OLEContainer methods

The main `OLEContainer` methods are:

`Showproperties`: (*No arguments*). Displays the document's properties dialog.

`Doverb`: Takes an integer argument which is the ID of a 'verb' to perform. This can either be a positive integer representing one of the application-specific verbs (see the `verbs` property), or one of the standard verbs:

- 0 Specifies the action that occurs when an end-user double-clicks the object in its container.
- 1 Instructs an object to show itself for editing or viewing.
- 2 Instructs an object to open itself for editing in a window separate from that of its container.
- 3 Causes an object to remove its user interface from the view. Applies only to objects that are activated in-place.
- 4 Activates an object in place, along with its full set of user-interface tools, including menus, toolbars, and other objects. If the object does not support in-place activation, gives domain error.
- 5 Activates an object in place without displaying tools, such as menus and toolbars, that end-users need to change the behavior or appearance of the object.
- 6 Used to tell objects to discard any undo state that they may be maintaining without deactivating the object.

`Refresh`: (*No arguments*). Causes the display to be updated to reflect any changes to the source document.

`Closedocument`: (*No arguments*). Closes the document, remembers its state, and disconnects from the server application. You can still read the `contents` property to retrieve the document contents.

Example

```

    ▽DEMO_OLEContainer;doc_class;contents;DEMO
[1]  A Sample function demonstrating use of the OLEContainer object
[2]  DEMO←'□' □NEW 'Window' ◇ DEMO.title←'OLE Example'
[3]  DEMO.OLE.New 'OLEContainer' ◇ DEMO.OLE.align←-1
[4]  DEMO.Show
[5]  A
[6]  A Set the file property to empty vector to put up selection dialog.
[7]  A This will ask the user to choose a document to embed or link
[8]  DEMO.OLE.file←''
[9]  A
[10] A See what was selected
[11] doc_class←DEMO.OLE.progid
[12] :If doc_class≡''
[13]     A No valid document selected - user probably cancelled dialog
[14]     DEMO.Close 1
[15]     :Return
[16] :EndIf
[17] 'Document of class ',doc_class,' loaded'
[18] A
[19] A Read the raw bytes which represent (in encoded form) the document.
[20] A In a real example, we might save this data to file and later use
[21] A it to re-create the document in another OLE container
[22] contents←DEMO.OLE.contents
[23] 'The document requires ',(⊖pcontents),' bytes of storage'
[24] A
[25] A Wait for the user to close the window
[26] 0 0ρ□WE DEMO
    ▽

```

Properties

align anchors aquaadjust autoactivate autodraw border caption children class color contents data docstate doublebuffered dragsource droptarget enabled events extent file handle maxsize methods minsize modified name opened order pointer progid properties scale self size sizemode sourceformats style tabstop targetformats tie units verbs visible where winptr

In addition, any properties exported by the embedded object will be available, with the name prefixed by 'x'. You can omit the prefix if the name does not clash with an APLX built-in name.

Methods

Click Clienttoscreen Close Closedocument Delete Doverb Draw Focus Hide New Open Paint Refresh Resize Screentoclient Send Set Show Showproperties Trigger

In addition, any methods exported by the embedded object will be available, with the name prefixed by 'X'. You can omit the prefix if the name does not clash with an APLX built-in name.

Callbacks

onClick onClose onDbIclick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave onDragOver onDragStart onFocus onHide onKeyDown onKeyPress onKeyUp onMouseDown onMouseMove onMouseUp onOpen onResize onSend onShow onUnFocus

In addition, any event callbacks defined by the embedded object will be available, with the name prefixed by 'onX'. You can omit the 'X' in the prefix if the name does not clash with an APLX built-in name.

OpenFile

Description

The `OpenFile` class implements the pre-defined file-selection dialog to allow the user to choose a file to open. This is a top-level object (i.e. it should not be opened as the child of a window).

To use the dialog, you must first create it using `NEW`. (Alternatively, you can create it as a top-level object using the `New` or `Create` method of `OWI`). It will remain hidden at this stage. You can then optionally set the initial file name/path as the `file` property. This can be either:

- A full path name.
- Just the file part, in which case you get the current working directory.
- Just a directory ending in a directory-separation character (`\` or `/` or `:` depending on the host system), in which case you get no initial file but the initial directory will be set.

The `style` property defines the behavior of the dialog. It is an integer (default 0), as follows:

```
1 = Allow multi-select
```

Under Windows and Linux, you can also add one or more of the following style codes (these are ignored under MacOS):

```
2 = Disable re-sizing
4 = Show 'Open as read-only' control
8 = Check 'Open As Read-Only'
16 = Path must exist
32 = File must exist
```

Under Windows and Linux, the `default` property defines the default file extension which will automatically be added if the user enters a filename without an extension. (*Not implemented under MacOS*).

The `filter` property sets one or more *filters* (file extensions or types which will be allowed). It comprises one or more sets of descriptive text, a `|` character, and a semi-colon delimited list of supported filters. Multiple filters should be separated by a vertical bar. Under Windows and Linux, you specify file extensions as file wildcards. For example, the filter `'Text files|*.txt;*.log|All files|*.*'` gives the user the choice of selecting 'Text files' (in which case extensions `.txt` and `.log` are accepted), or 'All files' (any extension is accepted).

Under MacOS, you can choose to filter either using file extensions (in which case the syntax is identical to that used for Windows and Linux), or using file types. In the latter case the filters are four-character file types rather than file extensions beginning with the wildcard `*.*`. For example: `'Picture files|JPEG;PICT'` would display the description "Picture files" to the user, and allow selection of files of type JPEG and PICT. The file type `'*****'` matches all file types. You can also

combine both types of filter; for example, 'Text files|*.txt;TEXT' would match files which either have the extension .txt or which are of type TEXT.

The `filterindex` property determines which of the filters (in index origin 1) is displayed from the drop-down list.

Displaying the dialog

When you are ready to show the dialog, call the `Show` method. This displays the modal dialog. The user can then select a directory; if the Cancel button is pressed, the `Show` method returns 0. If the OK button is pressed, the `Show` method returns 1. You can then read the `file` property to retrieve the file selected by the user. If the multi-select bit was on in the `style` property, this will be a nested vector of file names.

Finally, you will normally call the `Delete` method to dispose of the dialog. Alternatively, you can keep it for re-use later in your application.

Example

```

    ▽DEMO_OpenFile;VERSION;⍵IO;DLG
[1]  ⍺ Sample function demonstrating use of the OpenFile object
[2]  DLG←'⍵' ⍵NEW 'OpenFile'
[3]  ⍺
[4]  ⍺ Are we running on Windows/Mac/Linux?
[5]  ⍺ The way we specify the file filter differs
[6]  ⍵IO←1
[7]  VERSION←'⍵' ⍵WI 'version'
[8]  :If VERSION[2]≠0
[9]  ⍺ Running under Windows or Linux:
[10] ⍺ Set up default file
[11]   :If VERSION[2]=1
[12]     DLG.file←'c:\temp.txt'
[13]   :Else
[14]     DLG.file←'/temp.txt'
[15]   :EndIf
[16] ⍺
[17] ⍺ Set up filter
[18]   DLG.filter←'Text files (*.txt *.log)|*.txt;*.log|APL Workspaces (*.aws)|
    *.aws'
[19]   :Else
[20] ⍺ Running under MacOS:
[21] ⍺ Set up default file
[22]   DLG.file←'temp.txt'
[23] ⍺
[24] ⍺ Set up filter
[25]   DLG.filter←'Text files|TEXT|APL Workspaces|XWS'
[26]   :EndIf
[27] ⍺
[28] ⍺ Show the dialog. If user presses Cancel, quit
[29]   →(0=DLG.Show)/0
[30] ⍺ He pressed OK.
[31]   'File selected: ',DLG.file
    ▽

```

Properties

caption children class data default events file filter filterindex methods name opened properties self style tie

Methods

Close Create Delete New Open Send Set Show Trigger

Callbacks

onClose onDestroy onOpen onSend

Page

Description

The Page class implements the individual pages or tab sheets within a Selector object, like the 'Display' and 'Syntax Coloring' tabs in the APLX Preferences dialog. A Page must have a Selector as its parent, and typically will have other controls (such as check boxes and edit fields) as its children.

The Page `caption` property sets the title of the tab sheet.

The `order` property gives the position of the Page within the Selector, in index origin 1. You can change the position by writing a new value to `order`. Other Pages are re-numbered accordingly. Setting a non-integer value will place the Page between two existing pages.

For a Page object, the `onShow` and `onHide` callbacks are triggered when the page is selected/deselected.

Placing images on the tabs

Under Windows and Linux, you can optionally place images on the tabs of Page controls. To do this, you need to create an ImageList object which contains all the images which appear in the Pages of a given Selector object. You then write the name of the ImageList to the Selector object's `imagelist` property.

Finally, you set the `imageindex` property of each Page where you want to display an image. This selects one of the images from the ImageList and displays it on the tab. (See the description of the ImageList object for an example.)

Example

```

▽DEMO_Page;DEMO
[1]  ⍝ Sample function demonstrating use of the Selector object and pages
[2]  DEMO←'□' □NEW 'Dialog' ♦ DEMO.scale←1 ♦ DEMO.size←14 40
[3]  DEMO.title←'Selector/Page Example'
[4]  ⍝
[5]  ⍝ Create selector to hold the individual pages
[6]  ⍝ (Could just use "align _1" here to make selector same size as the window.
[7]  ⍝ However we make it slightly smaller to get the drop shadow under MacOS X)
[8]  DEMO.Sel.New 'Selector' ♦ DEMO.Sel.size←12 38 ♦ DEMO.Sel.anchors←1 1 1 1
[9]  ⍝
[10] ⍝ Now create three pages each with some radio buttons
[11] DEMO.Sel.Page1.New 'Page' ♦ DEMO.Sel.Page1.caption←'Appetizer'
[12] DEMO.Sel.Page1.R1.New 'Radio' ♦ DEMO.Sel.Page1.R1.caption←'Smoked Salmon'
[13] DEMO.Sel.Page1.R1.where←2 1 ♦ DEMO.Sel.Page1.R1.value←1
[14] DEMO.Sel.Page1.R2.New 'Radio' ♦ DEMO.Sel.Page1.R2.caption←'Fresh asparagus'
[15] DEMO.Sel.Page1.R2.where←4 1
[16] DEMO.Sel.Page1.R3.New 'Radio' ♦ DEMO.Sel.Page1.R3.caption←'Clam chowder'
[17] DEMO.Sel.Page1.R3.where←6 1
[18] ⍝

```

```

[19] DEMO.Sel.Page2.New 'Page' ◊ DEMO.Sel.Page2.caption←'Main course'
[20] DEMO.Sel.Page2.R1.New 'Radio' ◊ DEMO.Sel.Page2.R1.caption←'Roast duck'
[21] DEMO.Sel.Page2.R1.where←2 1 ◊ DEMO.Sel.Page2.R1.value←1
[22] DEMO.Sel.Page2.R2.New 'Radio' ◊ DEMO.Sel.Page2.R2.caption←'Grilled steak'
[23] DEMO.Sel.Page2.R2.where←4 1
[24] DEMO.Sel.Page2.R3.New 'Radio' ◊ DEMO.Sel.Page2.R3.caption←'Fresh sea-bass'
[25] DEMO.Sel.Page2.R3.where←6 1
[26] 
[27] DEMO.Sel.Page3.New 'Page' ◊ DEMO.Sel.Page3.caption←'Dessert'
[28] DEMO.Sel.Page3.R1.New 'Radio' ◊ DEMO.Sel.Page3.R1.caption←'Fresh
stawberries'
[29] DEMO.Sel.Page3.R1.where←2 1 ◊ DEMO.Sel.Page3.R1.value←1
[30] DEMO.Sel.Page3.R2.New 'Radio' ◊ DEMO.Sel.Page3.R2.caption←'Creme caramel'
[31] DEMO.Sel.Page3.R2.where←4 1
[32] DEMO.Sel.Page3.R3.New 'Radio' ◊ DEMO.Sel.Page3.R3.caption←'Baked Alaska'
[33] DEMO.Sel.Page3.R3.where←6 1
[34] 
[35]  Wait for the user to close the window
[36] 0 0pWE DEMO
  ▽

```

Properties

align anchors aquaadjust autodraw caption children class color data doublebuffered dragsource droptarget enabled events extent font handle imageindex maxsize methods minsize name opened order pointer properties scale self size sourceformats tabstop targetformats tie units visible where winptr

Methods

Click Clienttoscreen Close Create Delete Draw Focus Hide New Open Paint Resize Screentoclient Send Set Show Trigger

Callbacks

onClick onClose onDblClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave onDragOver onDragStart onFocus onHide onKeyDown onKeyPress onKeyUp onMouseDown onMouseMove onMouseUp onOpen onSend onShow onUnfocus

Picture

Description

The Picture class allows you to display various kinds of picture (for example, JPEG, Windows bitmaps, or MacOS PICT images). The object can optionally include scroll bars for scrolling around the image (if the image is bigger than the object), or can allow stretching of the image to fit the object size. The image can be read from file, from an APL array of pixel color values, or from a stored vector of drawing commands.

See also the `Draw` method which allows you to draw icons, pictures and bitmaps on your windows and controls.

Properties to specify the image

You can specify the image which is displayed in one of four ways:

- By writing the name of an image file (for example, 'mypic.bmp' or 'myphoto.jpg') to the `file` property. This can be a full pathname and filename, or just a file name in the current directory. (For compatibility with other APL systems, you can alternatively write the name to the `bitmap` property). The image file must be of a type supported by the host system on which you are running APLX. If you write an empty vector to the `file` property, a dialog is displayed allowing the user to select an image file.
- By writing an array of pixel color values to the `bitmap` property. The image is represented as an APL integer matrix, where each element contains an RGB color for the corresponding pixel. The color value is encoded as `2561Blue Green Red`, where each color value is in the range 0 to 255. You can also write a boolean array where 0 represents black and 1 represents white.
- By writing an integer vector containing drawing instructions (in the format appropriate to the host platform) to the `picture` property. Under MacOS, this is a PICT or JPEG object. Under Windows, it is a Windows Enhanced Metafile. In both cases, the data can for example be obtained from the clipboard (as the `picture` property of the System object). This option is not supported under Linux.
- By creating an Image object as a child of the Picture object. In this case, the Picture will display the current image from the off-screen bitmap of the Image object. When the Image object is changed, the Picture will update automatically.

Reading back the `bitmap` property always returns an APL array of color values, so you can use this to convert an image from file (such as a JPEG picture) to a bitmap.

Other properties

`style`: An integer comprising one of:

- 0 Picture is unscaled and centred
- 4 Picture is scaled to fit image
- 8 Picture is always scaled to fit frame, even when frame resized

to which can be added:

- 16 Vertical scroll bar (Not valid if `style = 8`)
- 64 Horizontal scroll bar (Not valid if `style = 8`)
- 128 Disable unnecessary scroll bars instead of removing them (MacOS only)

`imagesize`: The size (in current `scale` units) of the picture. This may be larger than or smaller than the `size` property of the object itself. If the `style` is 8, you cannot set `imagesize`, since it is always the same as the object size.

`bitmapsize`: The size (in current `scale` units) of the image within the picture.

Example

```

∇DEMO_Picture;DEMO
[1]  ⍝ Sample function demonstrating use of the Picture object
[2]  DEMO←'⍺' ⍵NEW 'Window' ⍵ DEMO.scale←5 ⍵ DEMO.size←188 140
[3]  DEMO.title←'Picture'
[4]  DEMO.myPicture.New 'Picture' ⍵ DEMO.myPicture.align←1
[5]  ⍝
[6]  ⍝ Specify the picture either from a PICT or JPEG file, e.g.
[7]  ⍝
[8]  ⍝ Macintosh:
[9]  ⍝   DEMO.myPicture.file←'Macintosh HD:MyPicture'
[10] ⍝
[11] ⍝ Windows:
[12] ⍝   DEMO.myPicture.file←'C:/MyFiles/MyPicture.jpg'
[13] ⍝
[14] ⍝ ...or put up a dialog asking the user to choose an image file:
[15] ⍝
[16] ⍝   DEMO.myPicture.file←''
[17] ⍝
[18] ⍝ ...or from an image held in a variable or the clipboard, e.g.
[19] ⍝
[20] ⍝   DEMO.myPicture.picture←('⍺' ⍵WI 'picture')
[21] ⍝
[22] ⍝
[23] ⍝ ...or from an integer matrix of encoded RGB colour values, as here:
[24] ⍝
[25] DEMO.myPicture.bitmap←SAMPLE_BITMAP
[26] ⍝
[27] ⍝ Wait for the user to close the window
[28] 0 0ρ⍵WE DEMO

```

∇

Properties

align anchors aquaadjust autodraw bitmap bitmapsizes border children class color data dragsource droptarget enabled events extent file imagesize maxsize methods minsize name opened picture pointer properties scale self size sourceformats style targetformats tie units visible where

Methods

Click Clienttoscreen Close Create Delete Draw Hide New Open Paint Resize Screentoclient Send Set Show Trigger

Callbacks

onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave onDragOver onDragStart onHide onMouseDown onMouseMove onMouseUp onOpen onSend onShow

Printer

Description

The `Printer` class gives you control over the printer, and also allows you to display the standard dialogs related to printing. This is an invisible, top-level object (i.e. it should not be created as a child of a window object).

To use a `Printer` object, you first create it as a top-level object. You can then if you wish call the `Setup` and `Job` methods to display the standard printer page-setup and print-job dialogs, allowing the user to set the printing parameters. Both of these methods return 1 if the user selected OK and 0 for Cancel.

Output to the printer or spool file begins when you `Open` the printer (this starts a new print job). You have control over the font and placement of the text using the `font` and `position` properties. Calling the `Print` method allows you to print text (either as a character vector, possibly with embedded carriage returns, or as a character matrix). Printing starts at the current position, and the `position` property is updated at the end of the `Print` method to reflect where the printing position has reached. If, during printing, the position reaches the bottom margin of the page, a page throw will automatically be generated and printing will resume at the top, left of the next page. You can fine-tune the line height and character pitch using the `lineheight` and `pitch` properties.

To draw graphics and arbitrary text on the printer page, you can use the `Draw` method. (If you have already used the `Draw` method to create the graphics in a window, you can retrieve a nested array of drawing commands from the window using the `Draw 'State'` keyword, and then re-submit the same commands to the `Printer` object as the argument of the printer's `Draw` method).

The `Eject` method forces a page throw. When you have output all the text, you call the `Close` method which ends the print job and submits it to the printer. If you want to abort the print job, call the `Abort` method.

The `margin` property, a four-element vector, allows you to set or retrieve the Left/Right/Top/Bottom margins in the current `scale`. Other useful properties include `copies` (the number of copies which will be printed), and `orientation` (a boolean scalar, 0=Portrait, 1=Landscape). These are read-only once the print job has started using `Open`.

There are a number of read-only properties associated with a `Printer` object. These include `page` (the current page number), `size` (the page size), `handle` (the handle to the printer record, used in low-level programming), and `winptr` (the display context/port also for low-level programming). Under Windows, the `fonts` property lists available printer fonts.

Note that, although it is possible to create more than one `Printer` object, only one can be used at a time. In general, we recommend that you do not create more than one `Printer` in your application.

Automatic (unattended) printing

If you want to print without user intervention (i.e. without displaying a dialog), you must set the parameters for the print job *before* calling the `Open` method. You can select the page orientation by setting the `orientation` property, and (except under MacOS), the number of copies by setting the `copies` property.

Under Windows only, if you have several printers accessible from your system, you can also select a particular one under program control. The `printers` read-only property returns a nested vector of the names of each available printer. The `printername` property allows you to select one of these, or query which one is selected.

Example

```

▽DEMO_Printer;TEXT;PRN
[1]  ⍝ Sample function demonstrating use of the Printer object
[2]  PRN←'⎵' ⎵NEW 'Printer' ⋄ PRN.scale←3 ⋄ PRN.margin←72 72 72
[3]  ⍝
[4]  ⍝ Put up the printer select dialog, quit on cancel
[5]  →(0=PRN.Setup)/0
[6]  ⍝
[7]  ⍝ (We could also display the Job dialog here)
[8]  ⍝
[9]  ⍝ Now open the printer to start the job, and print some text
[10] PRN.Open
[11] PRN.font←'Times' 36 1
[12] PRN.color←220
[13] PRN.Print('Quotation',⎵R)
[14] PRN.font←'Times' 18 0
[15] PRN.color←0
[16] PRN.Print('Let me not to the marriage of true minds',⎵R)
[17] PRN.Print('Admit impediments. Love is not love',⎵R)
[18] PRN.Print('Which alters when it alteration finds,',⎵R)
[19] PRN.Print('Or bends with the remover to remove.',⎵R,⎵R)
[20] PRN.font←'Times' 9 2
[21] PRN.Print 'William Shakespeare'
[22] ⍝
[23] ⍝ Close printer to submit the job
[24] PRN.Close
▽

```

Properties

caption children class color copies data events font fonts handle lineheight margin methods name opened orientation page pitch position printers printername properties scale self size tie units winptr

Methods

Abort Close Create Delete Draw Eject Job New Open Print Send Set Setup Trigger

Callbacks

onClose onDestroy onOpen onSend

Progress

Description

The Progress class implements the 'progress bar' control, typically used to display the progress of a long operation such as copying a large number of files.

Progress properties

The main Progress properties are:

`style`: The basic style is 0 for a vertical control. Add 1 to make it use smooth increments (recommended), and add 2 to make it a horizontal control (*not needed for MacOS - the orientation is deduced from the `where` property*).

`value`: Up to 4 integers of Position, Maximum, Step, Minimum.

`range`: A two-element integer vector giving the minimum and maximum values (default 0 100)

`increment`: An integer scalar giving the amount by which the value will change when the `StepIt` method is run. without an argument

Progress method

`StepIt`: Increment the value. If called with no arguments, the progress bar value is increased by the `increment` property. If called with an argument (integer scalar), it increments by the amount specified.

Example

```

▽DEMO_Progress;N;X;DEMO
[1]  ⍝ Sample function demonstrating use of the Progress object
[2]  ⍝ Default is progress bar from 0 to 100
[3]  DEMO←'□' □NEW 'Dialog' ♦ DEMO.title←'Progress Example'
[4]  DEMO.myProgress.New 'Progress' ♦ DEMO.myProgress.where←2 1
[5]  DEMO.myProgress.scale←1 ♦ DEMO.myProgress.style←1
[6]  ⍝
[7]  ⍝ Must show window now, otherwise it won't appear until after loop below
[8]  DEMO.Show
[9]  ⍝
[10] :For N :In 10
[11]   X←□DL 0.3
[12]   DEMO.myProgress.value←(N×10)
[13] :EndFor
[14] ⍝
[15] ⍝ Wait for the user to close the window
[16] 0 0□WE DEMO
▽

```

Properties

align anchors aquaadjust autodraw caption children class color data doublebuffered dragsource droptarget enabled events extent handle increment maxsize methods minsize name opened order pointer properties range scale self size sourceformats style tabstop targetformats tie tooltip units value visible where winptr

Methods

Click Clienttoscreen Close Create Delete Draw Focus Hide New Open Paint Resize Screentoclient Send Set Show Stepit Trigger

Callbacks

onClick onClose onDbIclick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave onDragOver onDragStart onFocus onHide onKeyDown onKeyPress onKeyUp onMouseDown onMouseMove onMouseUp onOpen onSend onShow onUnFocus

Radio

Alternative name: Option

Description

The Radio class implements 'radio' buttons, for selecting one-of-many mutually-exclusive options.

You typically set the `caption` property which indicates the caption associated with the control, and you might set the `enabled` property to 1 or 0 depending on whether a particular option is valid. The `value` property is 1 if the control is selected, and 0 otherwise. Only one radio button in a group can be selected at any time; selecting a different Radio button automatically de-selects the current one.

For Radio buttons, it is not always necessary to have any callbacks defined, since when the dialog ends or a Button is pressed you can determine the state of Radio objects using the `value` property. In other cases, use of an `onClick` callback to detect changes in the state of these controls may be preferable, and it is necessary if other items in a dialog need to depend on the state of the Radio or Check boxes.

The `group` property of a Radio button determines which group (within a particular control) the button belongs to. Only one Radio button in the group can be selected at any time, so that if one is selected, all others in the same group are automatically de-selected.

Usually, this grouping occurs automatically (typically, you place Radio buttons in a Frame object). The initial `group` property for the first Radio button to be created as the child of a Frame or Window is 0. If you create further Radio buttons one after another with the same parent, the `group` property for each new button is the same as that of the previous button, so each button in the series belongs to the same group.

If you create a *different* type of control in between, the `group` property is incremented by one for the next series, so that the second series of Radio buttons behaves independently of the first.

Alternatively, you can override this behavior and set the Radio button's `group` explicitly, as an integer scalar in the range 0 to 255.

To provide help text when the user moves the mouse pointer over the control and pauses, you can use the `tooltip` property.

Example

```
∇DEMO_Radio;DEMO
[1]  A Sample function demonstrating use of Radio buttons
[2]  DEMO←'□' □NEW 'Dialog' ♦ DEMO.scale←1
[3]  DEMO.title←'Radio Example'
```

```

[4]  #
[5]  DEMO.RADIO1.New 'Radio'
[6]  DEMO.RADIO1.where←2 3 1 30
[7]  DEMO.RADIO1.caption←' APLX for MacOS'
[8]  #
[9]  DEMO.RADIO2.New 'Radio'
[10] DEMO.RADIO2.where←3.5 3 1 30
[11] DEMO.RADIO2.caption←'APLX for Windows'
[12] #
[13] DEMO.RADIO3.New 'Radio'
[14] DEMO.RADIO3.where←5 3 1 30
[15] DEMO.RADIO3.caption←'APLX for Linux'
[16] DEMO.RADIO3.value←1
[17] #
[18] DEMO.RADIO4.New 'Radio'
[19] DEMO.RADIO4.where←6.5 3 1 30
[20] DEMO.RADIO4.caption←'APLX for Solaris'
[21] DEMO.RADIO4.enabled←0
[22] #
[23] # Wait for the user to close the window
[24] 0 0ρWE DEMO

```

▽

Properties

align anchors aquaadjust autodraw caption children class color data doublebuffered dragsource droptarget enabled events extent font group handle maxxsize methods minsize name opened order pointer properties scale self size sourceformats tabstop targetformats tie units value visible where winptr

Methods

Click Clienttoscreen Close Create Delete Draw Focus Hide New Open Paint Resize Screentoclient Send Set Show Trigger

Callbacks

onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave onDragOver onDragStart onFocus onHide onKeyDown onKeyPress onKeyUp onMouseDown onMouseMove onMouseUp onOpen onSend onShow onUnFocus

Rectangle

Description

The Rectangle object is normally used for display purposes only. The `where` or `size` property is used to define the coordinates of the rectangle.

You can set the `filled` property to indicate that you want the object filled with the foreground color. You can set the foreground color using the `color` property. (The background color is ignored). The `pensize` property changes the thickness of the lines.

Note: Remember that the default scale is in character units. See the `scale` property for details.

See also the `Draw` method which allows you to draw geometric shapes on your windows and controls.

Example

```

▽DEMO_Rectangle;DEMO
[1]  ⍠ Sample function demonstrating use of the Rectangle object
[2]  DEMO←'□' □NEW 'Window' ◇ DEMO.scale←1
[3]  DEMO.title←'Rectangle Example'
[4]  ⍠
[5]  DEMO.RECT.New 'Rectangle'
[6]  DEMO.RECT.where←4 4 3 6
[7]  DEMO.RECT.color←255
[8]  ⍠
[9]  DEMO.FILLEDRECT.New 'Rectangle'
[10] DEMO.FILLEDRECT.where←4 12 3 6
[11] DEMO.FILLEDRECT.color←(255×256)
[12] DEMO.FILLEDRECT.filled←1
[13] ⍠
[14] ⍠ Wait for the user to close the window
[15] 0 0ρ□WE DEMO
▽

```

Properties

align anchors aquaadjust autodraw children class color data dragsource droptarget enabled events extent filled maxsize methods minsize name opened pensize pointer properties scale self size sourceformats targetformats tie units visible where

Methods

Click Clienttoscreen Close Create Delete Hide New Open Paint Resize Screentoclient Send Set Show Trigger

Callbacks

onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave
onDragOver onDragStart onHide onMouseDown onMouseMove onMouseUp onOpen onSend
onShow

RichEdit

Description

The RichEdit class implements the RichEdit control, which is a text input and display control with advanced formatting capabilities, including support for multiple fonts, colors, and styles. Typically, you set the 'selection' properties such as `selcolor` and `selfont` to choose a text format, and then output some text using the `seltext` property. You can then change the format and output some more text. The RichEdit control can also be used for advanced word-processing type applications.

RichEdit properties

The main RichEdit properties are:

`style`: The style is a set of flags:

```

1 = centered
2 = right justified
4 = multiline
16 = vertical scroll bar
32 = inhibit auto horizontal scroll (ie word wrap if multiline)
64 = horizontal scroll bar
1024 = show selection even when control does not have focus (Windows only)
4096 = read only
8192 = allow Enter characters to be input (multi-line only)
65536 = disable unnecessary scroll bars rather than removing them
      (Windows and MacOS only)
131072 = allow Tab characters to be input, instead of tabbing to next control
      (Windows and MacOS only)

```

`text`: A character vector containing the text of the whole control. Writing to the `text` property replaces all the current text in the control. (Under Windows, you can also use the `rtf` property which is the same except that the text is in Rich Text Format and includes formatting information.)

`selection`: A two-element integer vector (*start*, *length*) giving the current selection start position (in index origin 0) and length. Writing to the `selection` property changes the selection on the screen, and determines where the next text will be output (i.e. the insertion point). A *length* of 0 means that no text is selected. A *length* of `-1` selects from the given *start* position to the end of the text.

`seltext`: A character vector containing the text of the selected part of the control. If there is no selection, this is an empty vector. Writing to the `seltext` property deletes any current selection and inserts the text you write, in the current selection font, style and color. (Under Windows, you can also use the `selrtf` property which is the same except that the text is in Rich Text Format and includes formatting information.)

`selcolor`: The current selection color, in the same format as the `color` property.

`selfont`: The current selection font and style, in the same format as the `font` property.

`selstyle`: The current selection style, as a 5-element vector: bold, italic, underline, strikeout, and protected. Valid values for each are 1 = Set, 0 = Clear, -1 = Leave unchanged. Setting the 'protected' attribute means the text cannot be changed, even under program control; it must therefore be set only after you have output text to the control. (*Note: The 'protected' attribute is not available under MacOS*).

`selalign`: The current paragraph alignment, as a character vector keyword. One of 'left', 'right', 'center' (or 'centre')

`selbullet`: The current paragraph bullet style, as a character vector keyword or boolean scalar. One of 'none' (or 0), 'bullet' (or 1)

`selindents`: A three-element numeric vector giving the left, bullet and right indents in the current scale of the control

`seltabs`: A numeric vector giving the tab positions in the current scale of the control. (*Windows only*)

`linecount`: (Read-only) The number of lines of text in the control

`range`: (Read-only) A two-element integer vector. The first element is the line number of the first visible line, and the second is the count of lines visible in the control.

RichEdit methods

The main RichEdit methods are:

`Clear`: Delete the currently-selected text.

`Cut`: Cut the currently-selected text to the clipboard.

`Copy`: Copy the currently-selected text to the clipboard.

`Paste`: Replace the currently-selected text with the contents of the clipboard.

`Undo`: Undo the last change.

`Print`: Print the contents of the control (with formatting), to the current printer. Can optionally be called with a character-vector argument, which is the print-job name. (You can use the `Setup` and `Job` dialogs in the `Printer` object to allow the user to configure the printer.)

`Save`: Saves the contents of the rich-edit control to a file. You can specify the file name as the argument, or specify an empty vector in which case a dialog is displayed to allow the user to specify the file name. You can also optionally pass a second argument, to indicate whether the saved file

should include formatting information. The valid values are 0 (no formatting, save as plain text), 1 (under Windows, save in Rich Text Format as a .rtf file), or 2 (under MacOS, save as Styled text).

Load: Loads the contents of a file into the rich-edit control. You can specify the file name as the argument, or specify an empty vector in which case a dialog is displayed to allow the user to choose a file name. Under Windows, the file can be in Rich Text Format (.rtf), in which case formatted text is loaded.

Linetochar: Takes a line number, and returns the position within the text of the first character of the line. Both are in index origin 0.

Chartoline: Takes a character position within the text, and returns the number of the line containing the character. Both are in index origin 0.

LineLength: Takes a line number (in index origin 0), and returns the length of the line in characters.

Example

```

▽DEMO_RichEdit;TEXT;DEMO
[1]  ⍝ Sample function demonstrating use of the RichEdit object
[2]  DEMO←'⍋' ⍋NEW 'Window' ⍋ DEMO.title←'RichEdit Example'
[3]  DEMO.myRichEdit.New 'RichEdit' ⍋ DEMO.myRichEdit.align←¯1
[4]  DEMO.myRichEdit.scale←3 ⍋ DEMO.myRichEdit.border←0
[5]  DEMO.myRichEdit.selindents←20 20 20
[6]  ⍝
[7]  DEMO.myRichEdit.selection←0
[8]  DEMO.myRichEdit.selfont←'Times New Roman' 18 1
[9]  DEMO.myRichEdit.selcolor←220
[10] DEMO.myRichEdit.seltext←('Quotation',⍋R)
[11] ⍝
[12] DEMO.myRichEdit.selfont←'Times New Roman' 12 0
[13] DEMO.myRichEdit.selcolor←0
[14] TEXT←⍋R,'Let me not to the marriage of true minds',⍋R
[15] TEXT←TEXT,'Admit impediments. Love is not love',⍋R
[16] TEXT←TEXT,'Which alters when it alteration finds,',⍋R
[17] TEXT←TEXT,'Or bends with the remover to remove.',⍋R,⍋R
[18] DEMO.myRichEdit.seltext←TEXT
[19] ⍝
[20] DEMO.myRichEdit.selfont←'Times New Roman' 10 2
[21] DEMO.myRichEdit.selalign←'right'
[22] DEMO.myRichEdit.seltext←'William Shakespeare'
[23] ⍝
[24] ⍝ Wait for the user to close the window
[25] 0 0ρ⍋WE DEMO
▽

```

Properties

align anchors aquaadjust autodraw border canundo caption children class color data doublebuffered dragsource droptarget enabled events extent font handle limit linecount maxsize methods minsize name opened order pointer properties rtf scale selalign selbullet selcolor selection self selfont selindents selrtf selstyle seltabs seltext size sourceformats style tabstop targetformats text tie units visible where winptr

Methods

Chartoline Clear Click Clienttoscreen Close Copy Create Cut Delete Draw Focus Hide Linelength
Linetochar Load New Open Paint Paste Print Resize Save Screentoclient Send Set Show Trigger Undo

Callbacks

onChange onClick onClose ondblclick ondestroy ondragdrop ondragend ondragenter
ondragleave ondragover ondragstart onfocus onhide onkeydown onkeypress onkeyup
onmousedown onmousemove onmouseup onopen onsend onshow onunfocus

RoundRect

Description

The RoundRect object implements a rectangle with rounded corners, and is normally used for display purposes only. The `where` or `size` property is used to define the coordinates of the rectangle. The `rounding` property determines the curvature of the corners.

You can set the `filled` property to indicate that you want the object filled with the foreground color. You can set the foreground color using the `color` property. (The background color is ignored). The `pensize` property changes the thickness of the lines.

Note: Remember that the default scale is in character units. See the `scale` property for details.

See also the `Draw` method which allows you to draw geometric shapes on your windows and controls.

Example

```

vDEMO_RoundRect;DEMO
[1]  a Sample function demonstrating use of the RoundRect object
[2]  DEMO←'□' □NEW 'Window' ♦ DEMO.scale←1
[3]  DEMO.title←'RoundRect Example'
[4]  a
[5]  DEMO.RECT.New 'RoundRect'
[6]  DEMO.RECT.where←4 4 3 6
[7]  DEMO.RECT.color←255
[8]  a
[9]  DEMO.FILLEDRECT.New 'RoundRect'
[10] DEMO.FILLEDRECT.where←4 12 3 6
[11] DEMO.FILLEDRECT.rounding←2
[12] DEMO.FILLEDRECT.color←(255×256)
[13] DEMO.FILLEDRECT.filled←1
[14] a
[15] a Wait for the user to close the window
[16] 0 0ρ□WE DEMO
v

```

Properties

align anchors aquaadjust autodraw children class color data dragsource droptarget enabled events extent filled maxsize methods minsize name opened pensize pointer properties rounding scale self size sourceformats targetformats tie units visible where

Methods

Click Clienttoscreen Close Create Delete Hide New Open Paint Resize Screentoclient Send Set Show Trigger

Callbacks

onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave
onDragOver onDragStart onHide onMouseDown onMouseMove onMouseUp onOpen onSend
onShow

SaveFile

Description

The `SaveFile` class implements the pre-defined file-selection dialog to allow the user to choose a name and location for saving a file. This is a top-level object (i.e. it should not be opened as the child of a window).

To use the dialog, you must first create it using `NEW`. (Alternatively, you can create it as a top-level object using the `New` or `Create` method of `OWI`). It will remain hidden at this stage. You can then optionally set the initial file name/path as the `file` property. This can be either:

- A full path name.
- Just the file part, in which case you get the current working directory.
- Just a directory ending in a directory-separation character (`\` or `:` depending on the host system), in which case you get no initial file but the initial directory will be set.

Under Windows, the `style` property defines the behavior of the dialog (it is ignored under MacOS). It is an integer (default 0) made up of the sum of:

```
2 = Disable re-sizing
4 = Warn if the selection would overwrite an existing file
8 = Give error if the user selects a read-only file (Windows only)
16 = Path must exist
```

Under Windows and Linux, the `default` property defines the default file extension which will automatically be added if the user enters a filename without an extension. It is not implemented in *APLX for MacOS*.

The `filter` property sets one or more *filters* (file extensions or types which will be allowed). It comprises one or more sets of descriptive text, a `|` character, and a semi-colon delimited list of supported filters. Multiple filters should be separated by a vertical bar.

Under Windows and Linux, you specify file extensions as file wildcards. For example, the filter `'Text files|*.txt;*.log|All files|*.*'` gives the user the choice of selecting "Text files" (in which case extensions `.txt` and `.log` are accepted), or "All files" (any extension is accepted).

Under MacOS, you can choose to filter either using file extensions (in which case the syntax is identical to that used for Windows and Linux), or using file types. In the latter case the filters are four-character file types rather than file extensions beginning with the wildcard `'*.'`. For example: `'Picture files|JPEG;PICT'` would display the description "Picture files" to the user, and allow selection of files of type `JPEG` and `PICT`. The file type `'*****'` matches all file types. You can also combine both types of filter; for example, `'Text files|*.txt;TEXT'` would match files which either have the extension `.txt` or which are of type `TEXT`.

The `filterindex` property determines which of the filters (in index origin 1) is displayed from the drop-down list.

Displaying the dialog

When you are ready to show the dialog, call the `Show` method. This displays the modal dialog. The user can then select a directory; if the Cancel button is pressed, the `Show` method returns 0. If the OK button is pressed, the `Show` method returns 1. You can then read the `file` property to retrieve the file selected by the user. This will be a character vector.

Example

```

    vDEMO_SaveFile;VERSION;⍵IO;DLG
[1]  ⍎ Sample function demonstrating use of the SaveFile object
[2]  DLG←'⍵' ⍵NEW 'SaveFile' ⍵ DLG.caption←'Save demonstration file'
[3]  ⍎
[4]  ⍎ Are we running on Windows or Mac? The way we specify the file filter
[5]  ⍎ is different...
[6]  ⍵IO←1
[7]  VERSION←'⍵' ⍵WI 'version'
[8]  :If VERSION[2]≠0
[9]    ⍎ Running under Windows or Linux:
[10]   ⍎ Set up default file
[11]   :If VERSION[2]=1
[12]     DLG.file←'c:\temp.txt'
[13]   :Else
[14]     DLG.file←'/temp.txt'
[15]   :EndIf
[16] ⍎
[17] ⍎ Set up filter
[18]   DLG.filter←'Text files (*.txt *.log)|*.txt;*.log|APL Workspaces (*.aws)|
*.aws'
[19] :Else
[20]   ⍎ Running under MacOS:
[21]   ⍎ Set up default file
[22]   DLG.file←'temp.txt'
[23]   ⍎
[24]   ⍎ Set up filter
[25]   DLG.filter←'Text files|TEXT|APL Workspaces|AXWS'
[26] :EndIf
[27] ⍎
[28] ⍎ Show the dialog. If user presses Cancel, quit
[29] →(0=DLG.Show)/0
[30] ⍎
[31] ⍎ He pressed OK.
[32] 'File selected: ',DLG.file
    v

```

Properties

caption children class data default events file filter filterindex methods name opened properties self style tie

Methods

Close Create Delete New Open Send Set Show Trigger

Callbacks

onClose onDestroy onOpen onSend

Scroll

Description

The Scroll class implements scroll bars, allowing the user to move a slider to scroll around data or select a value from a range.

The main property for a Scroll object is `value`. This is a five-element vector, as follows:

- [1] Current value (initially 0)
- [2] Maximum value (default 100)
- [3] 'Page' increment (default 1)
- [4] Minimum value (default 0)
- [5] (Reserved: 0)

When setting this property, you do not have to set all the elements.

The `style` property determines the orientation of the scroll bar. It is 0 for Vertical or 1 for Horizontal.

When the user changes the position of the slider, the `onChange` callback is triggered.

Note that you do not need to create your own scroll bars for List, Tree, Picture, Edit, RichEdit and Document objects; these will automatically include scroll bars where necessary, provided you have set the correct `style` property.

Example

```

▽DEMO_Scroll;CB_SCROLL;DEMO
[1]  # Sample function demonstrating use of the Scroll object
[2]  DEMO←'□' □NEW 'Dialog' ◇ DEMO.scale←1
[3]  DEMO.title←'Scroll Bar Example'
[4]  DEMO.Scr1.New 'Scroll' ◇ DEMO.Scr1.where←2 1
[5]  DEMO.Scr1.value←0 100 10 0 0
[6]  DEMO.Lab.New 'Label' ◇ DEMO.Lab.where←4 1
[7]  DEMO.Lab.caption←'Value: 0'
[8]  #
[9]  # Create callback function which will run when the user spins the control
[10] 0 0□FX 'CB_SCROLL;X' 'X←DEMO.Scr1.value' 'DEMO.Lab.caption←"Value ",⌘1↑X'
[11] DEMO.Scr1.onChange←'CB_SCROLL'
[12] #
[13] # Wait for the user to close the window
[14] 0 0□WE DEMO
▽

```

Properties

align anchors aquaadjust autodraw caption children class color data doublebuffered dragsource droptarget enabled events extent handle maxsize methods minsize name opened order pointer properties scale self size sourceformats style tabstop targetformats tie tooltip units value visible where winptr

Methods

Click Clienttoscreen Close Create Delete Draw Focus Hide New Open Paint Resize Screentoclient Send Set Show Trigger

Callbacks

onChange onClick onClose ondblclick ondestroy ondragdrop ondragend ondragenter ondragleave ondragover ondragstart onfocus onhide onkeydown onkeypress onkeyup onmousedown onmousemove onmouseup onopen onsend onshow onunfocus

Selector

Description

The Selector class implements a container for a set of pages or tab sheets, such as is used in the APLX Preferences dialog. A Selector must have Page objects as its children, on which typically other controls will be placed.

The Selector `style` property governs the appearance of the tab sheets under Windows. It can be either 0 (meaning use tabs to choose the pages), or 1 (use buttons). In addition, you can add one or both of 8 = Allow multiple lines of tabs, 16 = Ragged right. Under MacOS, the `style` property is ignored.

The `value` property sets or retrieves the index of the currently-selected Page, starting at 1. This is the same as the `order` property of the current Page.

The `font` property can be used to change the font which is used to display the names of the tabs. It is also the default font for any children of the tab sheets.

See the Page class for more information and an example.

Properties

align anchors aquaadjust autodraw caption children class color data doublebuffered dragsource droptarget enabled events extent font handle imagelist maxsize methods minsize name opened order pointer properties scale self size sourceformats style tabrows tabstop targetformats tie units value visible where winptr

Methods

Click Clienttoscreen Close Create Delete Draw Focus Hide New Open Paint Resize Screentoclient Send Set Show Trigger

Callbacks

onChange onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave onDragOver onDragStart onFocus onHide onKeyDown onKeyPress onKeyUp onMouseDown onMouseMove onMouseUp onOpen onSend onShow onUnFocus

SendMail

Description

The `SendMail` object allows your APLX applications to send e-mail messages, using the SMTP protocol supported by most Internet Service Providers (ISPs). It complements the `GetMail` object, which can be used to retrieve incoming e-mails in your APLX application.

You can use the `SendMail` object for a number of purposes. For example, you might have an APLX application which automatically analyses some data on a regular basis, and e-mails the results to a list of recipients. Or you might use it as part of an APL-based mailing list application. You can also use it to e-mail automatically-generated diagnostic reports to software-support staff.

Caution: The ability to send e-mails automatically from APLX is very powerful, but you should be very careful not to abuse it by violating privacy or sending out unsolicited 'spam' e-mails to people who do not want to receive them. Not only is this objectionable to many people, it is almost certainly not permitted by your Internet Service Provider and could result in your internet account being terminated. In addition, in many jurisdictions it may be illegal.

Connecting to the SMTP server

The `SendMail` object should be created as a top-level object, i.e. not as the child of a Window or other control. You then need set up the following properties, to define the address of the SMTP mail server you are using, your username and password (if required). This information should be available from your ISP or network administrator:

`host`: This is a character vector, to which you should assign the Internet address of the SMTP server. It can be specified either as a domain name (such as `'smtp.myisp.com'`), or directly as an internet node address (such as `'168.9.211.53'`).

`port`: You can use this property to change the port used by the SMTP connection. For most applications you should leave this set to the default value of 25.

`user`: The user name for the account, if required, should be assigned to this property. (*Not currently supported under MacOS.*)

`password`: The password, if required for this account, should be assigned to this property. Not all ISPs require a password for sending mail, in which case you can leave this property as an empty vector. (*Not currently supported under MacOS.*)

Next, you need to call the `Open` method, which establishes the connection with the server. (*Note: Whilst you have this connection open, you are tying up resources on the server. You should aim to close the connection as soon as you can; if you fail to do so, the SMTP server will eventually time out and close the connection automatically.*) The `Open` method returns an integer scalar, which indicates

whether the connection succeeded. The returned value will be one of 0 = OK, 1 = Could not find host, 2 = Authentication error (i.e. incorrect username or password), 3 = Other error.

For example:

```
SM←'□' □NEW 'SendMail'
SM.host←'smtp.supernet.com'
SM.user←'microapl'
SM.password←'sesame'
SM.Open
0
```

Preparing the e-mail

Before sending an e-mail, you need to set it up by assigning values to the following properties of the `SendMail` object:

`subject`: The subject of the e-mail, as a character vector.

`from`: The e-mail address of the sender of the e-mail, as a character vector.

`to`: The recipients of the e-mail, specified as a comma-delimited list of e-mail addresses.

`cc`: The list of e-mail addresses to which the message should be 'carbon-copied', specified as a comma-delimited list. Leave this as an empty vector if you do not want the e-mail to be copied to people than the main recipients.

`bcc`: The same as `cc`, except that the list of recipients will not be shown on the e-mail sent out ('blind carbon copy').

`replyto`: The e-mail address to which any reply to an e-mail should be sent. It is a simple character vector. If you do not specify this, the 'from' address will be used.

`body`: The main text of the e-mail, as a simple character vector with carriage return (□R) characters separating paragraphs.

`attachments`: A list of the names of the files (if any) which you want to attach to the e-mail. You should provide full paths for these. You can specify them either as a nested vector of vectors, or as a character matrix, or as a CR-delimited simple character vector. (If you read back the property, APLX always returns the nested vector form). Leave this property as an empty vector if there are no attachments.

Sending the e-mail

Once you have set up the properties of the e-mail, you send it by calling the `SendMessage` method (you must be connected to the SMTP server to do this). This method takes no argument; it returns an error code as follows: 0 = No error, 1 = Host not found, 2 = Authentication error, 3 = General or

comms error, 4 = Attachment not found, 5 = Incomplete header (e.g. 'to' property not set), 6 = Recipient not found.

For example, assuming you have created a `SendMail` object 'SM' and successfully connected to the server, the following APLX statements would send an e-mail, with an attachment, to MicroAPL:

```
SM.to←'microapl@microapl.co.uk'
SM.from←'bob3@yahoo.com'
SM.subject←'APLX feedback'
SM.attachments←'c:\docs\APLXReview.doc'
SM.body←'APLX is fantastic! See attached review.',⎕R,'Thanks. Bob'
errcode←SM.SendMessage
```

Sending another e-mail

Once you have called the `SendMessage` method to send the e-mail, you can then write new values to the various properties, and send another message. To reset all the properties to their default (empty) values before assigning new values, call the `Clear` method; this is usually recommended to avoid accidentally retaining some parts of the previous message (such as an attachment). In some cases, however, you may want to send out variants of the same message, in which you can simply change those properties which are different, without calling `Clear`.

Closing the connection

Finally, you should call the `Close` method to terminate the SMTP session and free resources on the mail server.

Error reporting and diagnostics

If an error occurs when calling the `SendMessage` method, an APL I/O ERROR will be generated. If an error occurs when the connection is being established using the `Open` method, it will return a non-zero result, as described above. In either case, you can use the `status` property to find out more about the error. This returns a two-element integer vector. The first element is 1 if the connection to the SMTP server is active, else 0. The second element is the latest error code returned by the underlying operating-system networking code. (See for example, the Windows documentation and include file 'winsock.h' for details on these error codes.)

Another useful diagnostic tool is the `serverreply` property. This returns a character vector containing the status message which the server sent back after a call to `SendMessage` or `Open`. Usually it will begin "+OK..." to indicate success, but if there is a problem (for example, if the mailbox is already locked) it will contain an error message. This can comprise several lines, separated by carriage returns.

Other

`timeout`: This is an integer scalar property, which allows you to set the timeout for mail operations, in milliseconds.

Example

```

vDEMO_SendMail;host;user;password;address;errcode;SM;IO
[1]  A Sample function demonstrating use of the SendMail object
[2]  A To run this function, you need an Internet account with
[3]  A an SMTP mail server, and you need to know the host name,
[4]  A user name and password (if required) for this account.
[5]  A
[6]  A Ask the user for the mail account details
[7]  M←'Enter mail server (SMTP) host name: ' ⋄ host←DDBR M
[8]  M←'Enter user name: ' ⋄ user←DDBR M
[9]  M←'Enter password (if required): ' ⋄ password←DDBR M
[10] M←'Enter your e-mail address: ' ⋄ address←DDBR M
[11] A
[12] A Create the SendMail object and set up connection details
[13] SM←'M' NEW 'SendMail'
[14] SM.host←host ⋄ SM.user←user ⋄ SM.password←password
[15] A
[16] A Open the connection to the SMTP server
[17] errcode←SM.Open
[18] :If errcode≠0
[19]   :Select errcode
[20]   :Case 1
[21]     'Could not find host'
[22]   :Case 2
[23]     'Authentication error'
[24]   :Else
[25]     'Comms error, error code = ',⌘1↓SM.status
[26]   :EndSelect
[27]   :Return
[28] :EndIf
[29] A
[30] A
[31] A Set up the e-mail properties. We'll send an email back to ourselves
[32] SM.to←address
[33] SM.from←address
[34] SM.subject←'Test message: Using the SendMail object'
[35] SM.body←('APLX is fantastic!',⌘R,'Thanks.')
[36] errcode←SM.SendMessage
[37] :If errcode≠0
[38]   'Error ',(⌘errcode),' - message not sent'
[39] :EndIf
[40] A
[41] A Close connection and clean up
[42] SM.Close
v

```

Properties

attachments bcc body cc children class data events from host methods name opened port properties
replyto self status subject tie timeout to user

Methods

Clear Close Create Delete New Open Send Sendmessage Set Trigger

Callbacks

onClose onDestroy onOpen onSend

Licensing (*Windows and Linux versions*)

In the Windows and Linux versions of APLX, the SendMail object is based on the Indy networking classes. The following notices apply to these versions:



Portions of this software are Copyright (c) 1993 - 2003, Chad Z. Hower (Kudzu) and the Indy Pit Crew - <http://www.IndyProject.org/>.

THIS SOFTWARE IS PROVIDED BY Chad Z. Hower (Kudzu) and the Indy Pit Crew "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Series

Description

The Series object is an invisible object which contains a series of data points to be displayed on a Chart object. It must always be the child of a Chart.

As well as containing the data which is to be displayed on the chart, it also has attributes for customizing the appearance of the series.

For full details, see the separate section on the Chart and Series Objects

Example

```

▽DEMO_Chart;labels;W
[1]  # Sample bar chart (See workspace 10 SAMPLESCHART for more examples)
[2]  W←'□' □NEW 'Window' ◇ W.title←'Beer Consumption' ◇ W.visible←3
[3]  W.Ch.New 'Chart' ◇ W.Ch.align←1 ◇ W.Ch.type←'bar'
[4]  W.Ch.title←'Thirsty people'
[5]  W.Ch.subtitle←'Beer sales per head of population'
[6]  W.Ch.note←'Source: The Economist Pocket World in Figures, 2004'
[7]  W.Ch.yaxislabel←'Retail Sales per head, litres'
[8]  W.Ch.s1.New 'Series' ◇ W.Ch.s1.color←374479
[9]  W.Ch.s1.values←85.8 76.7 73.9 69.9 69.7 66.6 64.7 64.5 49 40.2 38.6 31.7
[10] labels←'Czech Republic' 'Venezuela' 'Germany' 'Denmark' 'S.Africa'
[11] labels←labels,'Austria' 'US' 'Australia' 'Canada' 'Belgium' 'Japan' 'UK'
[12] W.Ch.xlabels←labels
[13] #
[14] # Wait for the user to close the window
[15] 0 0ρ□WE W
▽

```

Properties

caption children class closevalues color colormarker data enabled events fillmarker fillpattern
 highvalues linetype linewidth lowvalues marker methods name opened openvalues properties self tie
 type usealtscale values visible xvalues yvalues

Methods

Close Create Delete Hide New Open Send Set Show Trigger

Callbacks

onClose onDestroy onOpen onSend

Socket

Description

The Socket class allows you to use the low-level networking facilities provided by BSD or Windows sockets, for client-server applications. (See also the GetMail, SendMail and HTTPClient classes, which provide a higher-level interface for mail and HTTP protocols, and which are much easier to use.) It allows your APLX applications to exchange data with other processes, either on the local machine or over a network.

Usually, you will use Socket objects to allow APLX applications to communicate with non-APL processes. You can also use them to allow APLX applications to communicate with each other across a network.

For more details on using sockets, consult any of the standard documents on BSD or Windows socket programming.

Creating and using a socket as a client

The steps you need to carry out at the client end of the connection are as follows:

1. Create a Socket object, as a top-level object.
2. If necessary, set the `family` and `protocol` properties to specify the type of socket you want.
3. Call the `Open` method to establish the connection with the server. This takes two arguments. The first is the network address of the server, as a character string; for example, this might be 'localhost' (your own machine), or '192.100.200.1' (an internet address) or 'www.microapl.co.uk' (an internet domain name). The second argument is the port number. Port numbers are usually associated with specific type of network protocol, for example port 80 is used for HTTP (web access). The `Open` method returns a status code indicating whether it succeeded: 0 = OK, 1 = Could not find host, 2 = Authentication error, 3 = Other error. If the connection was established successfully, you can now exchange data with the server.
4. To send data to the server, call the `Send` method. This takes a character vector argument. The data is sent exactly as it, without any character translation (i.e. the character vector is treated as a sequence of raw bytes). The return code is 0 for success, or 3 if an error occurs (see the description of the `status` property below).
5. To receive data from the server, call the `Receive` method. This takes no arguments. It returns a character vector, which is the data received (as an untranslated sequence of raw bytes). The APL task will block until the data has been received, or the time specified in the `timeout` property has elapsed. *Note:* If you do not want the task to be blocked waiting for data, use the `onReceive` event to ensure that `Receive` is called only when there is data already waiting.
6. To end the session, call the `Close` method. This takes no arguments.

Creating and using a socket as a server

At the server end of a connection, things are usually a little more complex. This is because a server typically needs to be able to handle multiple client connections simultaneously, and must not get blocked communicating with one client when other clients are also trying to access the service. One way of handling this is to use multiple APLX tasks, with one task per client connection, plus one master task which waits for clients to connect. Another way is to use a single APLX task for everything, and use events to ensure that the task never gets blocked waiting for a response from a client.

The task which is waiting for clients to connect starts up first. It does the following:

1. Create a `Socket` object, as a top-level object.
2. Set the `port` property, to specify which port to listen to. If necessary, set the `family` and `protocol` properties to specify the type of socket you want.
3. Call the `Listen` method. This sets up the socket so that it is associated with the port.
4. You can now do one of two things. If the server task does not need to do anything until a client tries to connect, you can simply call the `Accept` method. The task will block, waiting for the next connection (or until the time specified by the `timeout` property has elapsed). Alternatively, you can set up an `onConnectRequest` callback, and call the `Accept` method only when a client tries to establish a connection.
5. If an request from a client to establish a new connection is successful, the `Accept` method returns a non-zero positive integer. This is the internal handle for a new underlying operating-system socket, which can be used to exchange data with the client. The `Accept` method will return 0 if it timed out, or `-1` if it could not establish the connection.
6. The server now needs to create a new `Socket` object (as a top-level object), which will be used to exchange data with the client which has just connected. The internal handle returned by the `Accept` method should be written to the `handle` property of the new `Socket` object. It will often be convenient to create this new socket in a new APLX task (see the description of the APL object), although this is not essential. Data exchange takes place using the `Send` and `Receive` methods, as already described for the client end of the connection. When the connection is no longer needed, it should be closed using the `Close` method, and deleted.
7. Meanwhile, the server loops back to step 4 and calls `Accept` again to wait for a new client connection, or waits for an `onConnectRequest` event.

Key properties of the Socket object

`family`: An integer scalar which specifies the address family of the socket. The default is 2, `AF_INET`, the address family used for the internet. It should be specified before calling the `Open` or `Listen` methods.

`type`: An integer scalar which specifies the socket type. It defaults to 1, `SOCK_STREAM`. It should be specified before calling the `Open` or `Listen` methods.

`protocol`: An integer scalar which specifies the protocol. It defaults to 0, the default for the address family.

`port`: An integer scalar which specifies the port number.

`timeout`: An integer scalar which sets the timeout (in milliseconds) before the `Accept` or `Receive` methods will return if no connection request or data is received. A value of `-1` means 'wait for ever'. A value of 0 means 'return immediately'.

`status`: *Read-only* Returns the connection status as a two-element vector. The first element is 1 if the socket is connected, else 0. The second element is the latest error number from the underlying operating-system socket.

`handle`: An integer scalar, which is the handle of the underlying operating-system handle. You can use this, in conjunction with `DNA`, to access socket functions not supported by the `Socket` object. You also need to write the result of the `Accept` method to this value at the server end, when you have created a new socket to accept an incoming client connection.

Key methods of the `Socket` object

`Open`: Open a new connection. (This method should be called from the client end of the connection only.) It takes two arguments. The first is the address of the server (usually an internet domain name or address, or `'localhost'` for your own machine). The second is the port number to connect on. The result is a status code indicating whether the connection succeeded: 0 = OK, 1 = Could not find host, 2 = Authentication error, 3 = Other error. (More details of any error are available from the `status` property).

`Listen`: Establish the socket as a server listener. This method takes no arguments. (It should be called on the server end of the connection only.) It returns the same error code as `Open`.

`Accept`: Wait for a client to try to connect. This method takes no arguments. (It should be called on the server end of the connection only.) It causes the task to block until a connection request is made, or the `timeout` value is reached. It returns the `handle` to use for the client connection, or 0 if it timed out or `-1` if it could not establish the connection.

`Receive`: Receive some data from the socket. This method takes no arguments. It causes the task to block until data is available, or the `timeout` value is reached. It returns a character vector which is the data received from the socket, as a raw sequence of bytes with no character translation. It returns an empty vector if it times out or an error occurs.

`Send`: Send some data to the socket. This method takes as an argument a character vector, which is the raw data to send to the other end of the connection. It returns 0 on success, else 3 if an error occurred.

`Close`: Disconnect and close the socket. This method takes no arguments. You always ensure the connection is closed cleanly by calling this method when the session is complete.

Callbacks of the Socket object

`onConnectRequest`: This event occurs on the server end, when you have called `Listen` and a client tries to connect. It indicates that you should now call the `Accept` method to accept the connection.

`onReceive`: This event occurs both on clients and servers. It indicates that the other end has sent some data, which you can now read using the `Receive` method.

`onDisconnect`: This event occurs both on clients and servers. It indicates that the other end has closed the connection. You should now call `Close`, and delete the socket.

Example

```

vDEMO_Socket;x;count;handle;cmd;errcode;ServerSock;Client1;Client2;SendSock
[1]  ⍝ Show how to use the Socket object. For this demo, we set up a
[2]  ⍝ Server and two client APL tasks, all on the local machine
[3]  ⍝ using port 10000 (which hopefully won't clash with anything)
[4]  Client1←'⍋' ⍋NEW 'APL' ⍋ Client1.wssize←100000 ⍋ Client1.where←0 0 10 20
[5]  Client2←'⍋' ⍋NEW 'APL' ⍋ Client2.wssize←100000 ⍋ Client2.where←0 35 10 20
[6]  Client1.Open
[7]  Client2.Open
[8]  ⍝
[9]  ⍝ Create sockets in each client APL task
[10] Client1.Execute "MySock←'⍋' ⍋NEW 'Socket'"
[11] Client2.Execute "MySock←'⍋' ⍋NEW 'Socket'"
[12] ⍝
[13] ⍝ Create a socket for the server, in our main task
[14] ServerSock←'⍋' ⍋NEW 'Socket'
[15] ServerSock.port←10000
[16] errcode←ServerSock.Listen
[17] :If errcode≠0
[18]   ⍝ Oops, something went wrong
[19]   'An error occurred in Listen, error code ',¯1↑ServerSock.status
[20]   :Return
[21] :EndIf
[22] ⍝
[23] ⍝ After a few seconds, get the clients to try to connect, and await data
[24] x←⍋DL 3
[25] cmd←"x←MySock.Open 'localhost' 10000 ⍋ 'Server sent: ',MySock.Receive"
[26] Client1.Execute("x←⍋DL 3 ⍋ 'Connecting..' ⍋ ",cmd)
[27] Client2.Execute("x←⍋DL 6 ⍋ 'Connecting..' ⍋ ",cmd)
[28] ⍝
[29] ⍝ Meanwhile, loop round accepting connections
[30] count←0
[31] 'Waiting for clients to connect...' ⍋ ⍋CC ' '
[32] :Repeat
[33]   handle←ServerSock.Accept
[34]   :If handle>0
[35]     ⍝ A client is trying to connect
[36]     ⍝ Establish new socket to talk to the client.
[37]     ⍝ We'll just transfer one packet, then close the connection
[38]     'Got connection request.' ⍋ ⍋CC ' '
[39]     count←count+1
[40]     SendSock←'⍋' ⍋NEW 'Socket'
[41]     SendSock.handle←handle
[42]     errcode←SendSock.Send('Hello, client task ',¯count)

```

```

[43]     SendSock.Close
[44]     SendSock.Delete
[45]     :ElseIf handle=-1
[46]         a Oops, something went wrong
[47]         'An error occurred in Accept, error code ',-1↑ServerSock.status
[48]         :Leave
[49]     :EndIf
[50] :Until count=2
[51] a
[52] a Wait, then kill the child tasks (this will also delete their sockets)
[53] x←DDL 4
[54] Client1.Close ◊ Client2.Close
[55] a
[56] a Clean up this end as well
[57] ServerSock.Close
    ▽

```

Properties

children class data events family handle methods name opened port properties protocol self status tie timeout type

Methods

Accept Close Create Delete Listen New Open Receive Send Set Trigger

Callbacks

onClose onConnectRequest onDestroy onDisconnect onOpen onReceive onSend

Spinner

Description

The Spinner class implements the Spinner control, a pair of up-down or left-right arrows for incrementing/decrementing a value.

The Spinner properties are:

`style`: 0=Vertical 1=Horizontal (*Not needed under MacOS - the `where` property determines the orientation*).

`value`: An integer giving the current value

`range`: A pair of integers giving the minimum and maximum values (default 0 100)

`increment`: The step by which the value property will change each time the user clicks one of the arrows

`wrap`: A boolean indicating if the value wraps round when the maximum/minimum is passed.

`tooltip`: Provides help text when the user moves the mouse pointer over the control and pauses.

The `onChange` callback is invoked when the user changes the value.

Example

```

▼DEMO_Spinner;CB_SPIN;DEMO
[1]  # Sample function demonstrating use of the Spinner object
[2]  DEMO←'□' □NEW 'Dialog' ♦ DEMO.scale←1
[3]  DEMO.title←'Spinner Example'
[4]  DEMO.Spin.New 'Spinner' ♦ DEMO.Spin.style←1 ♦ DEMO.Spin.where←2 1
[5]  DEMO.Spin.value←32 ♦ DEMO.Spin.range←0 50
[6]  DEMO.Lab.New 'Label' ♦ DEMO.Lab.where←4 1 ♦ DEMO.Lab.caption←'Value 32'
[7]  # Create callback function which will run when the user spins the control
[8]  0 0ρ□FX 'CB_SPIN;X' 'X←DEMO.Spin.value' 'DEMO.Lab.caption←''Value: ',X'
[9]  DEMO.Spin.onChange←'CB_SPIN'
[10] #
[11] # Wait for the user to close the window
[12] 0 0ρ□WE DEMO
▼

```

Properties

align anchors aquaadjust autodraw caption children class color data doublebuffered dragsource dropout target enabled events extent handle increment maxsize methods minsize name opened order

pointer properties range scale self size sourceformats style tabstop targetformats tie tooltip units value
visible where winptr wrap

Methods

Click Clienttoscreen Close Create Delete Draw Focus Hide New Open Paint Resize Screentoclient
Send Set Show Trigger

Callbacks

onChange onClick onClose ondblclick ondestroy ondragdrop ondragend ondragenter
ondragleave ondragover ondragstart onfocus onhide onkeydown onkeypress onkeyup
onmousedown onmousemove onmouseup onopen onsend onshow onunfocus

Splitter

Description

The `Splitter` class implements the Splitter control. This is a vertical or horizontal line control, which allows the user to change the relative sizes of the controls on a window. For example, you might have a list box and a text edit, and use a Splitter to allow the user to move the boundary between them.

To do this, you first set up the leftmost (or topmost) control or controls, using the `align` property to align to the left (or top). You then create the Splitter, defining its position with the `where` property. Then you create the rightmost (or bottom) control, setting its `align` property to fill the remaining client area of the parent, as shown in the example below.

If the Splitter `style` is 0 (the default), it has the normal bevelled appearance. If `style` is 1, it is not bevelled. (*This parameter is ignored under MacOS*).

The `limit` property is a numeric scalar, which sets the minimum size of the panes in the current `scale`. The default is 0.

Example

```

▽DEMO_Splitter;TEXT;DEMO
[1]  A Sample function demonstrating use of the Splitter object
[2]  A We set up a list box and multiline edit fields, aligned to the
[3]  A left in each case, with a splitter between them
[4]  DEMO←'□' □NEW 'Window' ♦ DEMO.scale←5
[5]  DEMO.title←'Splitter Example'
[6]  A
[7]  A Set up the list box
[8]  DEMO.List1.New 'List' ♦ DEMO.List1.align←2 ♦ DEMO.List1.size←50 100
[9]  DEMO.List1.list←□M
[10] A
[11] A Now create the splitter, just to the right of the list box
[12] DEMO.mySplitter.New 'Splitter' ♦ DEMO.mySplitter.where←0 102
[13] A
[14] A Now create right edit object, aligned to fill remainder of client area
[15] DEMO.Edit1.New 'Edit' ♦ DEMO.Edit1.style←36 ♦ DEMO.Edit1.align←~1
[16] DEMO.Edit1.color←0 0 0 225 255 255 ♦ DEMO.Edit1.border←0
[17] TEXT←'This is an example of a window with a splitter.'
[18] TEXT←TEXT,□R,□R,'Try re-sizing the window and moving the splitter around.'
[19] DEMO.Edit1.text←TEXT
[20] A
[21] A Wait for the user to close the window
[22] 0 0□WE DEMO
▽

```

Properties

align anchors aquaadjust autodraw caption children class color data dragsource droptarget enabled events extent handle limit maxsize methods minsize name opened order pointer properties scale self size sourceformats style targetformats tie units visible where

Methods

Click Clienttoscreen Close Create Delete Hide New Open Paint Resize Screentoclient Send Set Show Trigger

Callbacks

onChange onClick onClose ondblclick ondestroy ondragdrop ondragend ondragenter ondragleave ondragover ondragstart onfocus onhide onkeydown onkeypress onkeyup onmousedown onmousemove onmouseup onopen onsend onshow onunfocus

System

Description

The System class implements the System object, which is the ultimate parent for all other objects belonging to a particular APL task. It gives information about the APLX system and allows your APL applications to use the Clipboard.

In addition, the System object of a child task is used to communicate with its parent (see the `Signal` method), and to share data using 'delta' properties.

If you use the `OWI` syntax, you do not need to create the System object; it always exists, under the name '#'. You can alternatively use the name '□'. To use the System object using dot-notation, you need to get a reference to it using `□NEW` in the normal way:

```
Sys←'□' □NEW 'System'
```

The main System properties and methods are:

System properties

`scale`: The scale used to position windows on the screen. The System object `scale` is inherited by all windows (and thus their controls) which you subsequently create.

`pointer`: The default mouse pointer type, which will be displayed unless the mouse is over a control which explicitly sets a different cursor.

`text`: The contents of the Clipboard, as a text vector. You can read this property to retrieve any text in the Clipboard, and write to it to place text in the Clipboard. If the Clipboard does not contain text, an empty vector is returned.

`unicode`: The contents of the Clipboard, as an integer vector comprising Unicode values. You can read this property to retrieve any Unicode text in the Clipboard, and write to it to place Unicode text in the Clipboard (you can write either integers, or characters). If the Clipboard does not contain Unicode text, an empty vector is returned.

`contents`: The contents of the Clipboard, returned if possible as an APL array. For example, you can copy a range of cells from Excel to the Clipboard, and read it into APL as a numeric or nested array. You can also write to the `contents` property, in which case the data will be available in the clipboard for APLX or for other applications to read back.

`picture`: (*Windows and MacOS only*) The contents of the Clipboard, as a Picture. You can read this property to retrieve any picture in the Clipboard, and write to it to place a picture on the Clipboard. The data is represented as a numeric vector, which internally holds a PICT or JPEG object under MacOS, or a Windows Enhanced Metafile under Windows. If the Clipboard does not contain a picture

which is in or can be converted to PICT or Enhanced Metafile format, an empty vector is returned. (This property is not implemented under Linux - use the `bitmap` property instead.)

`bitmap`: The contents of the Clipboard, as a bitmap. You can read this property to retrieve any picture in the Clipboard, and write to it to place a picture on the Clipboard. The data is represented as an APL integer array of pixel color values. If the Clipboard does not contain a bitmap (or an image which can be converted to a bitmap), an empty matrix is returned.

`aplkeyboard`: An integer scalar. It is 1 if the APL keyboard is being used, and 0 if the default (non-APL) keyboard is in use. You can also set this under program control, with additional options for the APL keyboard layout. Setting the value to 1 means 'select whichever APL keyboard layout the user prefers'. Setting it to 2 selects the 'standard' APL keyboard layout, and 3 selects the 'unified' APL keyboard layout.

`aquaadjust`: A property which causes the size of certain controls to be enlarged slightly to take account of the MacOS X 'Aqua' appearance. This property is inherited by windows and controls which you create. It is ignored under Windows and Linux.

System read-only properties

`fonts`: A nested array of the names of screen fonts installed on the system.

`size`: The screen height and width in the current `scale`.

`workarea`: The rectangle within the screen size which available for application use, allowing for any task bar or application dock area reserved by the operating system. *Note: Under Linux, this property returns the whole screen size.*

`children`: A nested vector of the names of all the top-level objects (windows, pre-defined dialogs, etc) which you have created.

`version`: A five-element integer vector containing:

```
[1] Version number for the System Class sub-system
[2] GUI environment code 0=MacOS 1=Windows, 2=X-Windows, 4=Console only
[3] OS Family code. For MacOS: 0 = Classic, 1 = OSX
                        For Windows: 0 = 95/98/ME, 1 = NT/2000/XP/Vista
[4] OS Major version
[5] OS Minor version
```

`classes`: A nested vector of character vectors, giving the names of all the System Classes available in your APLX system.

`xclasses`: A nested matrix of the OCX/ActiveX classes installed on your computer. This will always be empty under MacOS and Linux.

`oleclasses`: A nested matrix of the OLE Server Application classes installed on your computer. This will always be empty under MacOS and Linux.

`oledoctype`s: A nested matrix of the OLE document types for which applications are installed on your computer. This will always be empty under MacOS and Linux.

`file`: A character vector, containing the names of the files (or the command-line under Windows or Linux) which you need to open or print in your packaged APLX application.

`action`: A character vector containing the keyword 'open' or 'print' depending on how the user started up your packaged APLX application. It can be an empty vector if no operation is required.

`directory`: A character vector containing the path from which the running program (either APLX itself, or your packaged application) was loaded.

`printers`: A nested vector giving the names of the printers available on the system.

`taskid`: A unique integer identifying the APL task.

`host`: If you are running a Client-Server implementation of APLX, and the task is running on a Server, this property will contain the name (or IP address) of the server system. It will be 'localhost' if the Client and the Server are running on the same machine. It will be an empty string if this is not a Client-Server configuration.

System methods

`Reset`: This method takes no arguments. It resets the System Class (`□WI`) sub-system to its initial state, deleting all objects.

`Signal`: Used by a child task to send a signal to its parent.

System events

`onAboutMenu`, `onPreferencesMenu` and `onQuitMenu`: These are triggered under MacOS to indicate that the user has selected a menu item from the system or application menu, which your packaged application needs to respond to.

`onOpen`: If you have packaged your APL workspace into a standalone application, this event is triggered under MacOS when the user drags a file icon on to your application icon once the application is running. The `file` and `action` properties indicate what files you need to open or print.

`onSignal`: Invoked when a parent task sends a signal to a child task.

Example

```

▽DEMO_System;OLDSCALE;Sys
[1]  ⍺ Sample function demonstrating use of the System object
[2]  ⍺
[3]  ⍺ -- First show use of ⍺WI syntax --
[4]  ⍺ Show the screen size in pixels
[5]  OLDSCALE←'⍺' ⍺WI 'scale'
[6]  '⍺' ⍺WI 'scale' 5
[7]  ''
[8]  'Screen size: ',⍺'⍺' ⍺WI 'size'
[9]  '⍺' ⍺WI 'scale' OLDSCALE
[10] ⍺
[11] ⍺ -- Now show use of dot-notation syntax --
[12] Sys←'⍺' ⍺NEW 'System'
[13] ⍺
[14] 'Program started from: ',Sys.directory
[15] 'You have ',(⍺pSys.fonts),' fonts installed on your machine.'
[16] ⍺
[17] ⍺ Read the contents of any text in the clipboard
[18] ''
[19] 'Text in the clipboard:'
[20] 60p'-'
[21] Sys.text
[22] 60p'-'
[23] ''
[24] ⍺
[25] ⍺ Set the clipboard contents
[26] Sys.text←'This was put in the clipboard by APLX'
[27] ⍺
[28] 'Now try pasting text from the clipboard into a window'
▽

```

Properties

action aplkeyboard aquaadjust bitmap children class classes color contents data directory events file fonts host methods name oleclasses oledotypes opened picture pointer printers properties scale self size taskid text tie unicode units version workarea xclasses

Methods

Reset Send Set Signal Trigger

Callbacks

onAboutMenu onOpen onPreferencesMenu onQuitMenu onSend onSignal

Timer

Description

The Timer class implements a periodic timer. This is an invisible object, which calls the `onTimer` callback at a fixed interval (provided the object's `enabled` property is 1). The time interval is set by the `interval` property, and is expressed in milliseconds.

A Timer object can be a top-level object (created directly using `NEW`, or the `New` or `Create` method of `OWI`). It can also be the child of the System object, of a Window or Form, or of any standard control. It cannot be the child of a pre-defined dialog or another Timer object.

Example

```

▽DEMO_Timer;CB_TIMER;DEMO
[1]  a Sample function demonstrating use of the Timer object
[2]  DEMO←'□' NEW 'Window'
[3]  DEMO.where←4 4 ◇ DEMO.scale←1 ◇ DEMO.size←7 30
[4]  DEMO.title←'Timer Example'
[5]  DEMO.myLabel.New 'Label' ◇ DEMO.myLabel.where←2 1
[6]  DEMO.myLabel.caption←''
[7]  a Set up 1-second timer which updates the label text
[8]  DEMO.myTimer.New 'Timer' ◇ DEMO.myTimer.interval←1000
[9]  a Create a little callback function which will run once a second
[10] 0 0p□FX 'CB_TIMER' 'DEMO.myLabel.caption←□TIME'
[11] DEMO.myTimer.onTimer←'CB_TIMER'
[12] a
[13] a Wait for the user to close the window
[14] 0 0p1 □WE DEMO
▽

```

Properties

children class data enabled events interval methods name opened properties self tie

Methods

Close Create Delete New Open Send Set Trigger

Callbacks

onClose onDestroy onOpen onSend onTimer

ToggleButton

Description

The `ToggleButton` class is similar to a `Button`, but allows you to specify one or more images (as bitmaps) which can be displayed on the button face, depending on its state. It can optionally also display a text caption. In addition, it has built-in capability to act as a toggle button (like a Check box), or as one of a group (like a Radio button).

Specifying the images

You specify the images displayed on the button using the `bitmap` property. This is an integer matrix, containing between 1 and 4 images laid out side-by-side. These correspond to:

1. The image displayed when the button is in its 'normal' (Up) state.
2. The image displayed when the button is disabled.
3. The image displayed during the time that the button is being clicked. If the button is acting as an Action button (i.e. its `group` property is 0), the 'normal' (Up) image reappears when the user releases the mouse button.
4. The image displayed when the button is in its 'selected' (Down) state. This applies only if the button is used as a radio or toggle button.

If you provide fewer than 4 images, the `ToggleButton` will represent the other states automatically by altering the image slightly.

The actual images are specified as an APL integer matrix, where each element contains an RGB color for the corresponding pixel. The color value is encoded as `256⊖Blue Green Red`, where each color value is in the range 0 to 255. If you want to supply more than one image, these should be laid out horizontally; you specify the `imagecount` property to indicate how the array should be divided up. For example, suppose you supply an array 20 pixels high and 60 pixels wide. If you set `imagecount` to 1, the whole array will be treated as a single image of size 20 by 60, and will be used when the button is in its 'normal' (Up) state; images for the remaining states will be simulated automatically. If you set `imagecount` to 2, the array will be treated as representing two images side-by-side, each 20 by 30 pixels, representing the button in its 'normal' and disabled states respectively. If you set `imagecount` to 3, it will be treated as representing three images, each of 20 by 20 pixels, corresponding to the first three states listed above.

Depending on the `style` property, the bitmap may be transparent. If so, the pixel value in the lower left corner of each image will be used as the transparent color value; any other pixel with the same value will not be displayed, so the background shows through.

Determining the behavior of the button

The `group` property is an integer scalar which determines the behavior of the button. If it is 0 (the default), the button behaves like an ordinary 'action' button; when the user clicks on it, it temporarily displays the third image, and when the user releases the mouse it reverts to the normal ('up') state. You would typically use this by hooking it to the `onClick` callback to take some action when the button is pressed.

If the `group` property is non-zero, the button can remain selected after the user has clicked it. In this state, it displays the fourth image, and its `value` property becomes 1 rather than 0. However, it also interacts with other ToolButtons with the same `group` property; only one button in the group can be selected at a time, so clicking on one de-selects all the others. In addition, you can specify whether all the buttons in the group can be deselected, using the `style` property (see below). If the `style` property contains 8, it is legal for none of the buttons in the group to be selected; if it does not contain 8, exactly one of the buttons in the group will be selected at all times.

Thus, you can use the button in three main ways:

- *As an 'Action' button:* Leave the `group` property as 0, and use the `onClick` event to detect when it is pressed. It never stays selected.
- *As a 'Toggle' button:* Set the `group` property to a unique non-zero value, so the button does not interact with any other buttons. When the user clicks on the button, it will change to the 'selected' (Down) state (`value = 1`). When the user clicks on it again, it will revert to the 'normal' (Up) state (`value = 0`). For this to work, you must ensure that the `style` property contains 8, otherwise the button can never be deselected.
- *As a 'Radio' button:* Set the `group` property to a non-zero value which is common to several ToolButtons with the same parent. When the user clicks on the button, it will change to the 'selected' (Down) state (`value = 1`), and any button in the group which is currently selected will be de-selected. If the `style` property contains 8, the user can click on the currently-selected button to de-select it, so that no buttons in the group are selected. Otherwise, exactly one of the buttons in the group will always be selected.

In the second and third of these cases, you can optionally use the `onClick` event to detect transitions, or you can simply read the `value` property when you need it.

Other key properties

`style`: An integer scalar comprising one of:

- 0 Caption (if any) appears below image (default)
- 1 Caption (if any) appears above image
- 2 Caption (if any) appears to left of image
- 3 Caption (if any) appears to right of image

to which can be added:

- 4 If this bit is set, the button appears 'flat', without a border. Recommended for toolbars.
- 8 Specifies that all buttons in a group can be deselected.
- 16 Specifies that the button is transparent

caption: A character vector, which specifies a caption to be displayed on the button. It defaults to an empty vector, meaning no caption is displayed. The location of the caption relative to the image is determined by the `style` property.

value: A boolean scalar indicating whether the button is selected. You can set this to select a button under program control.

tooltip: A character vector which gets displayed as help text when the user moves the mouse pointer over the control and pauses. You normally use this to provide information about what the button does. (It will be suppressed if you set the `tooltipenabled` property of the system object to 0).

enabled: A boolean scalar. If you set this to 0, the button will be disabled, will not respond to clicks, and cannot be selected. It will display the second of the images you supply, or a greyed-out image if you supply one.

Example

```

vDEMO_ToolButton;NORMAL;DISABLED;CLICKED;SELECTED;X;STATE;DEMO
[1]  A Sample function demonstrating use of the Tool Button object
[2]  DEMO←'□' □NEW 'Dialog' ◇ DEMO.scale←5 ◇ DEMO.size←60 235
[3]  DEMO.title←'ToolButton Example'
[4]  STATE←'Unselected' 'Selected'
[5]  A
[6]  A Create a simple 20 by 20 bitmap of a triangle shape, which we use to make
[7]  A four images for up/down/left/right, in four colors
[8]  X←(ϕX),X←20 10↑0,[1]2/[1]⇒9↑“(ι9)ρ”1
[9]  A
[10] A Choose some colors for our simple images.
[11] NORMAL←256ι0 255 0      A Green
[12] DISABLED←256ι160 160 255  A Pale Red
[13] CLICKED←256ι170 240 170  A Pale green
[14] SELECTED←256ι255 0 0      A Blue
[15] A
[16] A Make four tool buttons, with triangle in different color for each state
[17] A Style 0+4+8+16 for caption below image, 'flat', allow all up, transparent
[18] A Create the buttons in a radio group, so only one can be down at any time.
[19] DEMO.UP.New 'ToolButton'
[20] DEMO.UP.tooltip←'Up direction (disabled)' ◇ DEMO.UP.caption←'Up'
[21] DEMO.UP.where←0 0 50 40 ◇ DEMO.UP.style←28
[22] DEMO.UP.imagecount←4 ◇ DEMO.UP.group←99
[23] DEMO.UP.bitmap←((X×NORMAL),(X×DISABLED),(X×CLICKED),(X×SELECTED))
[24] DEMO.UP.onClick←"□←'Up Hit!'"
[25] A Show effect of disabling the Up button; we'll get the second (red) image
[26] DEMO.UP.enabled←0
[27] A
[28] A
[29] X←⊖X
[30] DEMO.DN.New 'ToolButton'
[31] DEMO.DN.tooltip←'Down direction' ◇ DEMO.DN.caption←'Down'
[32] DEMO.DN.where←0 40 50 40 ◇ DEMO.DN.style←28
[33] DEMO.DN.imagecount←4 ◇ DEMO.DN.group←99
[34] DEMO.DN.bitmap←((X×NORMAL),(X×DISABLED),(X×CLICKED),(X×SELECTED))
[35] DEMO.DN.onClick←"□←'Down Hit! ',STATE[□I0+□WSELF □WI 'value']"
[36] A
[37] X←⊚X

```

```

[38] DEMO.RT.New 'ToolButton'
[39] DEMO.RT.tooltip←'Right direction' ⋄ DEMO.RT.caption←'Right'
[40] DEMO.RT.where←0 80 50 40 ⋄ DEMO.RT.style←28
[41] DEMO.RT.imagecount←4 ⋄ DEMO.RT.group←99
[42] DEMO.RT.bitmap←((X×NORMAL),(X×DISABLED),(X×CLICKED),(X×SELECTED))
[43] DEMO.RT.onClick←"□←'Right Hit!',STATE[□IO+□WSELF □WI 'value']"
[44]  ␣
[45]  X←φX
[46] DEMO.LF.New 'ToolButton'
[47] DEMO.LF.tooltip←'Left direction' ⋄ DEMO.LF.caption←'Left'
[48] DEMO.LF.where←0 120 50 40 ⋄ DEMO.LF.style←28
[49] DEMO.LF.imagecount←4 ⋄ DEMO.LF.group←99
[50] DEMO.LF.bitmap←((X×NORMAL),(X×DISABLED),(X×CLICKED),(X×SELECTED))
[51] DEMO.LF.onClick←"□←'Left Hit!',STATE[□IO+□WSELF □WI 'value']"
[52]  ␣
[53]  ␣ Wait for the user to close the window
[54]  0 0ρ□WE DEMO

```

▽

Properties

align anchors aquaadjust autodraw bitmap caption children class color data dragsource droptarget
 enabled events extent font group imagecount maxsize methods minsize name opened pointer
 properties scale self size sourceformats style targetformats tie tooltip units value visible where

Methods

Click Clienttoscreen Close Create Delete Hide New Open Paint Resize Screenshotclient Send Set Show
 Trigger

Callbacks

onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave
 onDragOver onDragStart onHide onMouseDown onMouseMove onMouseUp onOpen onSend
 onShow

Trackbar

Description

The Trackbar class implements a control which allows the user to move a slider to select a value. It is somewhat like a scroll bar in functionality, although the appearance is different. Under Windows, it also allows the programmer to display a current selection range, and to define the position of the tick marks along the control.

Trackbar properties

The main Trackbar properties are:

`style`: The basic style is 0 for Vertical and 1 for Horizontal. Under Windows, you can also add one of more of:

```
0 = Ticks on bottom or right side.  
2 = Ticks on top or left side.  
4 = Ticks on both sides.  
8 = Hide tick marks but display slider (Windows only).
```

`value`: The current slider position, as an integer scalar.

`range`: A two-element integer vector giving the minimum and maximum values (default 0 100)

`selection`: A two-element integer vector giving the 'selection' start and end, or an empty vector if there is no selection. This property is ignored under MacOS and Linux.

`increment`: A two-element vector giving the amount by which the value will change when the Page Up/Down keys are pressed, and the amount by which the value will change when the user clicks in the track or presses an arrow key. This property is ignored under MacOS because keyboard operation of the Trackbar is not supported.

`sliderlen`: The length of the slider in current `scale` units, or 0 to make it invisible. This property is ignored under MacOS and Linux.

`tickinterval`: The interval at which ticks appear, or 0 if just at Min and Max, or hidden. This property is ignored under MacOS.

`ticks`: An integer vector of user-defined tick positions (overrides `tickinterval`). This property is ignored under MacOS and Linux.

`tickpos`: A read-only matrix giving the values and positions of the tick marks. This property is not available under MacOS and Linux.

`sliderwhere`: A read-only property returning the position of the slider. This property is not available under MacOS and Linux.

The `onChange` callback is invoked when the user moves the slider.

Example

```

    ▽DEMO_Trackbar;VERSION;ΠIO;CB_TRACK;CB_CLOSE;X;DEMO
[1]  ▽ Sample function demonstrating use of the Trackbar object
[2]  ▽
[3]  ▽ The windows version of this function demonstrates features not
[4]  ▽ available on the Mac or Linux. The Mac/Linux Trackbar is very simple.
[5]  ΠIO←1
[6]  VERSION←'Π' ΠWI 'version'
[7]  :If VERSION[2]=1
[8]    ▽ Running under Windows:
[9]    DEMO←'Π' ΠNEW 'Dialog' ◇ DEMO.title←'Trackbar Example' ◇ DEMO.scale←1
[10]   DEMO.TBar.New 'Trackbar' ◇ DEMO.TBar.where←2 1
[11]   DEMO.TBar.style←1 ◇ DEMO.TBar.value←35
[12]   DEMO.Label1.New 'Label' ◇ DEMO.Label1.where←1 1
[13]   DEMO.Label1.caption←'Move slider to set threshold'
[14]   DEMO.Label2.New 'Label' ◇ DEMO.Label2.where←6 1 ◇ DEMO.Label2.color←255
[15]   DEMO.Label2.caption←'
[16]   ▽
[17]   ▽ Create a callback which will run when the user closes the window
[18]   ▽ This prevents the window being closed asynchronously
[19]   DEMO.onClose←'→'
[20]   ▽
[21]   ▽ Must show window now, otherwise it won't appear until after loop below
[22]   DEMO.Show
[23]   ▽
[24]   :While 1
[25]     ▽ Loop round until the CB_CLOSE callback has run
[26]     ▽ Set some random value in the 'selection' property
[27]     DEMO.TBar.selection←(0,?80)
[28]     X←ΠWE 0.3 ▽ Process events for up to 300 ms
[29]     :If (¬1↑DEMO.TBar.selection)<DEMO.TBar.value
[30]       DEMO.Label2.caption←'
[31]     :Else
[32]       DEMO.Label2.caption←'Threshold exceeded'
[33]     :End
[34]   :End
[35] :Else
[36]   ▽ Running under MacOS or Linux:
[37]   DEMO←'Π' ΠNEW 'Dialog' ◇ DEMO.title←'Trackbar Example' ◇ DEMO.scale←1
[38]   DEMO.TBar.New 'Trackbar' ◇ DEMO.TBar.where←2 1
[39]   DEMO.TBar.style←1 ◇ DEMO.TBar.value←35
[40]   DEMO.Lab.New 'Label' ◇ DEMO.Lab.where←4 1
[41]   DEMO.Lab.caption←'Value: 35'
[42]   ▽
[43]   ▽
[44]   ▽ Create a callback function which will run when user spins the control
[45]   0 0ρΠFX 'CB_TRACK;X' 'X←DEMO.TBar.value' 'DEMO.Lab.caption←"Value ",#1↑X'
[46]   DEMO.TBar.onChange←'CB_TRACK'
[47]   ▽
[48]   ▽ Wait for the user to close the window
[49]   0 0ρΠWE DEMO
[50] :EndIf
    ▽

```

Properties

align anchors aquaadjust autodraw caption children class color data doublebuffered dragsource droptarget enabled events extent handle increment maxsize methods minsize name opened order pointer properties range scale selection self size sliderlen sourceformats style tabstop targetformats tickinterval tickpos ticks tie tooltip units value visible where winptr

Methods

Click Clienttoscreen Close Create Delete Draw Focus Hide New Open Paint Resize Screentoclient Send Set Show Trigger

Callbacks

onChange onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave onDragOver onDragStart onFocus onHide onKeyDown onKeyPress onKeyUp onMouseDown onMouseMove onMouseUp onOpen onSend onShow onUnFocus

Tree

Description

The Tree class implements a control for displaying and manipulating hierarchical lists. It comprises a tree of 'nodes', which can be expanded or collapsed by the user. Examples are the left-hand pane of the APLX 'Workspace Explorer' window, or Microsoft's 'Windows Explorer'. Under Windows and Linux, you can optionally associate an ImageList with the Tree, to display images next to the nodes.

Tree properties

The main Tree properties are:

`style`: The sum of the following flags:

- 1 = Hide the buttons at the nodes.
- 2 = Hide the lines connecting the children.
- 4 = Show the top-level (root) node
- 8 = Highlight the selection even if control does not have focus
- 16 = Allow the user to edit the labels (Windows only)
- 128 = Disable rather than hide unneeded scroll bars (MacOS only)

`list`: A nested matrix (or vector) of either 3, 4 or 5 columns, and one row for each node. The columns are:

- [;1] An integer node identifier. If you specify 0, APLX assigns a unique value for each node.
- [;2] An integer representing node depth, typically 1 = the top level.
- [;3] A character vector giving the label to display for this node
- [;4] A vector of four indices into the list of images associated with the control. These are the Unselected, Selected, Overlay, and State images, in index origin 1. 0 means no image. Ignored under MacOS.
- [;5] A nested vector of keywords giving the state of the node.

The state keywords are: 'highlightbold', 'highlightcut', 'highlightdrop', 'highlightselect', 'expanded', 'expandedonce', 'haschildren' and 'forceparent'. Precede the keyword by a minus sign to switch off the state. The 'highlightcut', 'highlightdrop' and 'forceparent' keywords apply to Windows only, and are ignored under MacOS.

`imagelist` (*Windows and Linux only*): A character vector containing the name of the main ImageList associated with the Tree. The first two elements of the fourth column of the `list` property are indexes into this list of images, and the third element is an index into the `overlays` property of this list.

`imagelistuser` (*Windows and Linux only*): A character vector containing the name of the secondary (user) ImageList associated with the Tree. The fourth element of the fourth column of the `list` property is an index into this list of images.

`indent`: The tree indentation in the current `scale` units

`value`: The ID of the currently-selected node

`highlightbold`: An integer vector specifying which nodes are bold highlighted

`highlightcut`: An integer vector specifying which nodes are highlighted for cutting

`highlightdrop`: An integer vector specifying which nodes are highlighted for dropping

`highlightselect`: An integer vector specifying which nodes are highlighted for selection

Tree methods

The main Tree methods are:

`Expand`: Control how a node displays. The method takes a length-2 nested vector. The first element is a keyword indicating the action. The second element is an integer, the node ID. The action keywords are:

<code>expand</code>	Open node to display children
<code>collapse</code>	Close node to hide children
<code>toggle</code>	Expand closed node or collapse open node
<code>expandbranch</code>	Open node to display children (recursively)
<code>collapsebranch</code>	Close node to hide children (recursively)
<code>collapsereset</code>	Removes all descendants

`EnsureVisible`: Ensure that a given node is visible, expanding any parents if they are currently collapsed, and scrolling the node into view. Takes a single integer argument, the node ID.

`ShowNode`: Select a node, or make it visible as the first item visible in the window. The method takes a length-2 nested vector. The first element is a keyword indicating the action. The second element is an integer, the node ID, or is 0 or 'none' to indicate the currently-selected node. The action keywords are:

<code>select</code>	Select the node
<code>drop</code>	make the node the drag-and-drop target (Windows only)
<code>firstvisible</code>	Make node visible at top of window

`Ensurevisible`: Ensure that a given node is visible, expanding and/or scrolling the tree if necessary. The argument is the node ID to be shown.

`SortChildren`: (*Windows Only*) Sort the children of a node, and optionally recurses through all descendants. First argument is node identifier, or character vector 'root'. The optional second argument is the character vector 'recurse'.

`DeleteNodes`: Delete one or more nodes. Argument is a vector of node IDs to delete.

`InsertNodes`: Insert one or more nodes. The first argument is node id of parent (or 'root'). The second argument is node id of preceding sibling (or 'first', 'last', or 'sort'). The third argument is a matrix (or vector for single node), as per the `list` property.

`Setinfo`: Sets various attributes of one or more nodes.

`Getinfo`: Retrieves various attributes of one or more nodes (Windows only).

`Beginlabeledit`: Allows the user to start editing a node's label (Windows only).

`Cancellabeledit`: Ends label editing.

`Hittest`: Determines which node a point on the control corresponds to.

`Findnode`: Finds a node given various criteria.

Example

```

vDEMO_Tree;TREE;□IO;NAMES;DEMO
[1]  a Sample function demonstrating use of the Tree object
[2]  □IO←1
[3]  DEMO←'□' □NEW 'Window' ♦ DEMO.title←'Tree Example'
[4]  DEMO.myTree.New 'Tree' ♦ DEMO.myTree.align←1 ♦ DEMO.myTree.style←12
[5]  TREE←11 5ρ0
[6]  TREE[;2]←1 2 2 3 3 3 4 4 2 2 1
[7]  NAMES←'Work' 'APL Workspaces' 'Documents' 'Reports' 'Letters'
[8]  NAMES←NAMES,'Planning' 'Product plan' 'Strategic plan'
[9]  NAMES←NAMES,'Spreadsheets' 'Graphics' 'Personal'
[10] TREE[;3]←NAMES
[11] TREE[;4]←11ρ<1 1 0 0
[12] TREE[;5]←11ρ<0ρ' ' '
[13] TREE[1 3 6 11;5]←4ρ<'HIGHLIGHTBOLD' 'EXPANDED'
[14] DEMO.myTree.list←TREE
[15]  a
[16]  a Wait for the user to close the window
[17]  0 0ρ□WE DEMO
v

```

Properties

align anchors aquaadjust autodraw border caption children class color count data doublebuffered dragsource droptarget enabled events extent font handle highlightbold highlightcut highlightdrop highlightselect imagelist imagelistuser indent labeledithwnd list maxsize methods minsize name opened order pointer properties scale searchstring self size sourceformats style tabstop targetformats tie tooltip units value visible where winptr

Methods

Beginlabeledit Cancellabeledit Click Clienttoscreen Close Create Delete Deletenodes Draw Ensurevisible Expand Findnode Getinfo Focus Hide Hittest Insertnodes New Open Paint Resize Screentoclient Send Set Setinfo Show Shownode Sortchildren Trigger

Callbacks

onClick onClose onDbClick onDestroy onDragDrop onDragEnd onDragEnter onDragLeave
onDragOver onDragStart onFocus onHide onKeyDown onKeyPress onKeyUp onMouseDown
onMouseMove onMouseUp onOpen onSelChange onSend onShow onUnFocus

Section 3. System Classes: Properties

The 'action' property

Read-only, Character vector

Valid for: System object

The `action` property is useful only for APL workspaces which have been packaged as standalone applications. When one or more documents have been dragged on to the icon which represents your packaged application, or documents associated with your packaged application have been double-clicked, the operating system starts up the packaged application and runs your APL code. (Under MacOS, the System object's `onOpen` event is triggered if your application is already running). You can read the `file` property to find the list of documents (files) to open or print. Under MacOS, the `action` property can contain the string 'open' or 'print', to indicate the action to take.

Under Windows and Linux, this property is implemented, but the action string can only be 'open' since printing from the desktop is not implemented.

This property returns an empty vector if no action is required.

The 'activecell' property

Two-element integer vector

Valid for: Grid

The Row and Column of the currently-active cell. You can write to this property to change the current cell.

The 'align' property

Integer scalar

Valid for: All visible controls

The `align` property can be used to force a control to be aligned along the whole of one of the edges of its parent, or to fill the whole contents area of its parent. It overrides the `where` property, and forces the control to be positioned hard against the edge. It also forces it to expand or contract to be the same height or width as the parent. You can set it to one of five values:

- 0 No alignment (default)
- 1 The control is aligned along the top edge of its parent
- 2 The control is aligned along the left edge of its parent
- 3 The control is aligned along the bottom edge of its parent
- 4 The control is aligned along the right edge of its parent
- 1 The control fills the whole client area of its parent

For example, if you wanted a Browser control to fill the whole window, you could write:

```
Win1←'□' □NEW 'Window' ♦ Win1.where←4 4 14 28
Win1.WebCtl.New 'Browser' ♦ Win1.WebCtl.align←-1
```

The 'allowselection' property

Cell property of Grid object

Each cell element is a Boolean scalar

For each cell for which this property is 1 (default), the cell can be selected. If it is set to 0, the cell cannot be selected.

The 'anchors' property

Four-element boolean vector

Valid for: All visible controls

The `anchors` property defines which edges of the window the control is anchored to. It comprises a vector of four boolean values, which govern whether the control is anchored to the top, left, bottom and right of the window respectively. The default is `1 1 0 0`, meaning that the control is anchored to the top and left of the window. As the window is resized, the position of the control remains fixed relative to the top left.

If you set the `anchors` property to `0 0 1 1`, the control will be anchored to the bottom and right of the window. This means that, as the window is resized, the control will remain at a constant distance from the bottom right corner of the window (its `where` property will change accordingly), as in this example:

```
Win1.But.anchors←0 0 1 1
```

If a control is anchored to both the left and the right, then as the window is re-sized, the left edge of the control will remain at a fixed distance (as initially set by the `where` property) from the left edge of the window, and the right edge of the control will remain at a fixed distance from the right edge of the window. Thus the control will grow or shrink horizontally with the window. Similarly, if the control is anchored to both the top and the bottom, it will grow or shrink vertically with the window. This example causes the control to grow or shrink with the window in both directions:

```
Win1.Ed.anchors←1 1 1 1
```

The 'angle' property

A two-element integer vector

Valid for: Arc

The first element of the `angle` property determines the starting angle of the arc, measured in degrees with 0 representing vertically upwards. The positive direction is clockwise. The second element is the angle subtended by the arc, again in degrees. The special case of `0 0` is the default, and represents a complete circle or ellipse.

The 'aplkeyboard' property

Integer scalar

Valid for: System object

The `aplkeyboard` property is an integer scalar which determines whether the APL keyboard mapping is in effect. Setting this property to 1 causes input to all APLX windows (including those created using `OPEN` or `OWI`) to use the user's preferred APL keyboard layout (unified or standard). Setting the value to 2 selects the 'standard' APL keyboard layout, and 3 selects the 'unified' APL keyboard layout. Setting it to 0 causes the ordinary non-APL layout to be used. This is particularly useful for packaged applications, where you typically want to switch off the APL keyboard layout.

When you read the value back, you always get either 0 (for non-APL layout) or 1 (for APL layout).

The 'aquaadjust' property

Boolean scalar

Valid for: System, Window, Form, Dialog, Button, Check, Radio

The new look-and-feel (known as 'Aqua') which Apple introduced in MacOS X can sometimes require controls to be slightly larger than they were in the 'classic' MacOS environment. For example, Buttons have a more rounded look, which for some dialogs can mean that text which used to fit in the button is now partially obscured. This can be a problem if you have existing APLX or APL.68000 code which you want to move to MacOS X. Obviously, you can go through your code and adjust the sizes of the controls on each dialog, but this can be very time-consuming. The `aquaadjust` property provides a quick method of performing the adjustment automatically. If you set the `aquaadjust` property for the System object to 1, all subsequent Buttons, Radio buttons and Check boxes which are created will be adjusted in size slightly. The current value of this property is inherited at creation time by window objects, as well as the individual controls, so you can set it on or off individually for particular dialogs or controls.

The exact effect of setting `aquaadjust` to 1 is that Check and Radio controls will be sized to a minimum of 18 by 18 pixels, and Buttons will be increased in size by 12 pixels. The property must be set before you set the `size` or `where` property. Note that reading back the `size` or `where` property will return the actual size after the adjustment.

You may still have to do some manual adjustment even when `aquaadjust` is set, but it should reduce the work considerably.

The `aquaadjust` property is ignored under Windows and Linux.

The 'attachments' property

Nested vector of character vectors, or a simple character vector or matrix

Valid for: GetMail, SendMail

The `attachments` property contains the list of file names which correspond to the attachments of a mail message. For the GetMail object, it is a read-only property, which is valid only after you have called the `GetMessage` method to retrieve the message together with the attachments. (The files are written to the directory specified in the `path` property, and the names are returned without the directory part). For a SendMail object, you should write to this property to set up any attachments before calling the `SendMessage` method. For this case, you should specify the full file path.

You can specify this property either as a nested vector of file names, or as a carriage-return delimited simple character vector, or as a character matrix with one name per row (trailing blanks are stripped off). When you read the property, APLX always returns the nested vector form.

The 'autoactivate' property

Integer scalar

Valid for: OLEContainer (*Implemented under Windows only*)

The `autoactivate` property determines when a document in an OLEContainer control is 'activated' (i.e. made visible and, usually, editable). The possible values are:

```
0 = Manual (use the DoVerb method to activate the document)
1 = Automatic (APLX activates it automatically)
2 = Activate when the control gets focus
3 = Activate when the user double-clicks in the control.
```

The default is 1 - activation occurs automatically.

The 'autodraw' property

Boolean scalar

Valid for: All visible controls and windows

The `autodraw` property is a boolean scalar which determines whether commands issued using the `Draw` method are stored and automatically replayed when the window needs updating. The default is true (1).

If `autodraw` is 1, each time the window needs updating (for example, if you have resized it or uncovered it), the `Draw` commands stored with the object are re-run to update the window. This means that, once you have issued one or more `Draw` commands, you do not need to handle update events in your APL code.

Occasionally, however, you may not want this to happen, and you may want to handle re-drawing yourself. In this case you can set `autodraw` to 0, and use the `onPaint` callback to program the re-drawing in APL.

Any `Draw` commands issued while `autodraw` is set to 0 are not entered into the list of saved drawing commands associated with the object.

The 'autoeditstart' property

Integer scalar

Valid for: Grid

Determines whether the data in the Grid can be edited, and if so how the edit process starts. It can be one of:

- 0 Editing of a cell is possible only when the program calls `Editstart`
- 1 Editing is automatically enabled when the cell is selected; the user can immediately type
- 2 The user must press Enter (or, under Windows and Linux, F2) to start editing a cell

The default is 1.

The 'axiswidth' property

Integer Scalar

Valid for: Chart

The `axiswidth` property allows you to specify the line width (in pixels) for the Chart object's axis, as an integer scalar. The default is `-1`, which means use the value set in the Chart object's `linewidth` property.

The 'background' property

Boolean scalar

Valid for: APL (child task) object

By default, any child task you create using the `APL` object class will have its own session window. However, if you set the `background` property to `1`, the task will not have a session window and all its output will be thrown away (you must set this property before calling the `Open` method).

The 'barwidth' property

Integer Scalar in range 0 - 100

Valid for: Chart

The `barwidth` property allow you to specify the width of the bars used in bar charts. It is an integer scalar, in the range 0 to 100, representing the percentage of the available width used to draw the bars at each X value. (For a Bar or Horizontal Bar chart with multiple series, where the bars for each series are placed side by side, this width is shared equally between the different series).

If the `barwidth` property is set to 100, then 100% of the available space is used, so there will be no gaps between the bars. If it is set to a low value (say 20), then the bars will be very narrow, with lots of white space between them. The default is 80.

The 'bcc' property

Character Vector

Valid for: SendMail

The `bcc` ('blind carbon copy') property contains a list of e-mail addresses to which the message should be copied, without the other recipients knowing that these addresses have been sent a copy. The recipients are specified as a comma-delimited list of e-mail addresses. The default is an empty vector.

You should set this property before calling the `SendMessage` method.

The 'bitmap' property

Integer matrix (*also Character vector for Picture object*)

Valid for: Chart, Form, Image, Picture, ToolButton, System object, Window

Specifying or reading an array of pixel values

The `bitmap` property can be used to represent an image as an APL integer matrix, where each element contains an RGB color for the corresponding pixel. The color value is encoded as `256⊔Blue Green Red` where each color value is in the range 0 to 255. You can also specify a boolean array, in which case 0 is interpreted as black and 1 as white.

For the `Picture` object, writing to the `bitmap` property causes the bitmap to be displayed. Reading it returns the current picture as an APL array of colors.

For the `System` object, writing to the `bitmap` property places the bitmap on the Clipboard, ready for pasting into another application. Reading the `bitmap` property returns any bitmap currently on the Clipboard as an APL array of color values. If there is no bitmap on the Clipboard, an empty array is returned.

For an `Image` object, the `bitmap` property can be used to create the image, or to retrieve it as an array of color values.

For a `Chart`, `Form`, or `Window`, the `bitmap` property is read-only. It returns an array of color values which correspond to the displayed object. (*Under Linux, this property is not currently implemented for the Form or Window object. Under MacOS 9, it is valid for a Form or Window only if the window is completely visible on screen.*)

For a `ToolButton` object, the `bitmap` property can contain between 1 and 4 images, laid out side by side. These are displayed on the button, depending on its state. The `imagecount` property determines how many images the bitmap holds.

Specifying a file name (*Picture object only*)

You can also use the `bitmap` property to specify the name of a file containing an image which will be displayed in a `Picture` object. The file can be a Windows bitmap (.bmp), JPEG picture (.jpg or .jpeg), or, on the Macintosh, a PICT image or any other file for which a Quicktime graphics converter is installed. Under Linux, the supported types depend on the operating system version but will typically include .png .xpm .jpg .jpeg .ico and .bmp files. Reading back the `bitmap` property always returns an APL array containing the pixels in the picture, so you can use this property to convert one of the supported image types to an APL bitmap array.

The 'bitmapsizes' property

Read-only, two-element numeric vector

Valid for: `Picture`, `Image`

The `bitmapsizes` property contains the height and width of the image in a `Picture` or `Image` object, in the current `scale` units. For a `Picture`, this may be either larger or smaller than the visible area of the picture.

The 'body' property

Character Vector

Valid for: `GetMail`, `SendMail`

The `body` property contains the main text of an e-mail, as plain text. It is a simple character vector, with carriage return (␣) characters separating paragraphs.

For the `SendMail` object, you should set this property before calling the `SendMessage` method.

For the `GetMail` object, this property is read-only, and is valid only after you have called the `GetMessage` method to retrieve an e-mail from the POP3 server. If the e-mail message was created correctly by the program which sent it, this property should always contain the plain text of the message. However, some e-mail programs, and many senders of 'spam' (junk) mail, place HTML formatted text in this field, or leave it blank. If the message was sent with formatted (HTML) text, the formatted version should be available in the `html` property.

The 'border' property

An integer scalar

Valid for: Chart, Grid, List, Edit, RichEdit, OLEContainer, Picture, Tree, Form, Window, Document, Dialog

For a Chart, List, Edit, RichEdit, OLEContainer, Picture, or Tree, the `border` property is either 0 or 1. If it is 1, a frame is drawn around the object in the object's foreground color. The default is 0 for a Chart, Picture, Icon, OLEContainer, and 1 for a Grid, List, Edit, RichEdit or Tree.

For a top-level object, the border property may be set only at creation time (as part of the `New` or `Create` method). The value is made up of a basic border style, plus optional flags. The basic border styles are:

0	No border
1	Standard document window, window not resizable
2	Resizable document window
3	Modal-dialog style window
4	Movable modal-dialog style window
5	Thin title-bar, non-resizable window
6	Thin title-bar, resizable window

The optional flags which you can add to these basic values are:

16	Window includes a Close box
64	Window includes a Zoom box (under MacOS) or Minimize/Maximize controls (under Windows and Linux)

You cannot include a Close box if the basic `border` type is 0, and you cannot include a Zoom box if it is 0, 5 or 6.

The default values are 20 for a Dialog object, and $2+16+64 = 82$ for a Document or Window object.

Note that the setting of this property does not determine whether a given window operates in a modal way, but merely sets the appearance and sizing controls.

Under Linux, the request which you make for a particular style of border is passed to the Window Manager, which may choose not to honor it.

The 'borderstyle' property

Integer scalar

Valid for: Grid

The `borderstyle` property is the sum of a set of flags which determine the appearance of the Grid (the defaults are all On):

- 1 Header cells have separator horizontal line
- 2 Header cells have separator vertical line
- 4 Data cells have separator horizontal line
- 8 Data cells have separator vertical line
- 16 Use 3D effect (Windows and MacOS only, ignored under Linux)

The 'canundo' property

Read-only, Boolean scalar

Valid for: Edit, RichEdit, Document

The `canundo` property indicates whether the most recent change to the text in an edit control can be undone by calling the `Undo` method.

The 'caption' property

Alternative name: title

Character vector

Valid for: Window, Form, Dialog, Document, Browser, Chart, Label, Menu, Check, Radio, Button, ToolButton, Combo, Page, Frame, Chart, Printer (*Windows only*), MsgBox, ChooseDir, OpenFile, SaveFile

The `caption` property is the text displayed by a text control such as a static text, button, radio or check object, and the title of a menu, pre-defined dialog, or window. For a Printer object under Windows, it is the name of the print job which will display in the printer queue. It defaults to the name of the object when it is created. For example:

Create a button called OK on the window Win1. Its caption will be 'OK':

```
Win1.OK←'□' □NEW 'Button'
```

Change its caption to 'Yes':

```
Win1.OK.caption←'Yes'
```

Note: Because of a limitation in Windows, you cannot change the title of the ChooseFont and ChooseColor pre-defined dialogs.

For a Chart object, the `caption` property is one of three labels (the others are `subtitle` and `note`). It is intended as a title for the whole chart. However, you can use it in any way you wish, or omit it altogether by leaving it as an empty vector. You can specify the position, color and font of the displayed text by using the `placetitle`, `colortitle` and `fonttitle` properties.

For a Browser object, the `caption` property is read-only. It corresponds to the HTML Title tag in the page header, so should normally represent the title of the web-page currently displayed.

The 'cc' property

Character Vector

Valid for: GetMail, SendMail

The `cc` ('carbon copy') property contains a list of e-mail addresses to which the message is copied, with the other recipients knowing that these addresses have been sent a copy. The recipients are specified as a comma-delimited list of e-mail addresses.

For the SendMail object, you should set this property before calling the `SendMessage` method. The default is an empty vector.

For the GetMail object, this property is read-only, and is valid only after you have called the `GetMessage` method to retrieve an e-mail from the POP3 server.

The 'children' property

Read-only, Nested vector of names

Valid for: All objects

The `children` property contains a list of the children for an object. It returns a nested vector of the (fully-qualified) names of the children. Thus, the `children` property of the System object will return a list of all the top-level objects which you have created (such as windows, pre-defined dialogs, and printers). The `children` property of a window will return the name of the controls in the window. It is a read-only property.

The 'class' property

Read-only, Character vector

Valid for: All objects

The `class` property contains the class of the object, for example 'Button' or 'Timer'. It is the same as the class name used when you created the object using the `New` method.

The 'classes' property

Read-only, Nested vector of character vectors

Valid for: System object

The `classes` property of the System object returns a nested vector containing the names of each of the APLX object classes which you can use. Where alternative names are available for a particular class (such as 'Form and 'Window'), only the first name is returned.

The 'closevalues' property

Numeric Vector

Valid for: Series

The `closevalues` property sets the data used for the closing values of a Series object displayed on a High-Low-Open-Close or Candlestick chart. It is a numeric vector.

You should normally provide the same number of values for each of the `openvalues`, `closevalues`, `highvalues` and `lowvalues` properties. However, the `openvalues` and `closevalues` properties are optional, and can be left as an empty vector.

Note: The `closevalues` property is actually a synonym for `yvalues`, and contains the same data.

The 'col' property

Integer Scalar

Valid for: Grid

The Column number (in index origin 1) of the currently-active cell. You can write to this property to change the current cell.

The 'color' property

(Alternative name: `colour`)

Numeric vector - see description

Valid for: All displayable objects, Printer and Series

The `color` (or, if you prefer, `colour`) property determines the foreground and optionally the background color for an object. Colors are set by specifying the amount of Red, Green and Blue in the range 0 to 255. (Because of limitations in display hardware, the color displayed will not be exact but will be the closest available from the current palette.)

Colors can be specified in one of two ways. In the first way, you specify a vector of three integers (Red, Green and Blue), so that for example 255 0 0 is pure red and 255 255 255 is white. In the second way, you specify a single integer which encodes the three values in base 256 as 256 * Blue Green Red. In addition, two special values are recognised; -1 means use the parent's foreground or background color as appropriate, and -2 means use the default color for that object (usually black for foreground and white for background). The default value for most objects is -1 -1.

You can specify any of the following lengths for the value of the color property:

- 1 The foreground color encoded as a single integer.
- 2 The foreground and background colors both encoded as single integers
- 3 The foreground color as R G B values separately
- 6 Both foreground and background as R G B separately.

Referencing this property always returns a two-element vector of the encoded foreground and background colors.

For a Grid object, the foreground color you set determines the default color of the text shown in the Grid (you can change this for individual cells using the `colortext` cell property). The background color is used as the background for the data part of the Grid (you can change this for individual cells using the `colorback` cell property).

For a Series object, the `color` property determines the color used to display that series on a chart. Only the first (foreground) color applies. By default, this is chosen automatically when the series is created, but you can change it if you want to specify a particular color for that series. It is ignored if the `monochrome` property of the parent Chart object is 1.

Note: Colors for controls like buttons should be used with discretion. In general, User Interface Guidelines advise against using non-default colors for controls, and where these are used they should normally not be bright hues which attract too much attention and detract from the typical look of the screen. In addition, some controls will display in an ugly way if non-default colors are used. This is particularly true of background colors, which generally paint the enclosing rectangle of the object. A few controls will override the color which you set - for example, a color icon displays in its own colors, ignoring the foreground color you specify except for drawing the frame (if any) around the icon, and the color you set is ignored for scroll bars and pop-up menus (except for the frame, if any).

The 'coloraxis' property

(Alternative name: `colouraxis`)

Integer Scalar or Length-3 Vector

Valid for: Chart

This property allow you to set a specific color for the axes (and associated tick marks and labels) of the chart. It is specified as a single color value. It can be expressed as a vector of three separate RGB values each in the range 0 - 255, or as a single integer which encodes the three values in base 256 as $256 \times \text{Blue} + \text{Green} + \text{Red}$. The default is the special value `-1`, meaning use the Chart object's foreground color.

When you read back this property, APLX always returns the single-integer encoded form.

This property is ignored if the Chart's `monochrome` property is set to 1.

The 'colorback' property

(Alternative name: `colourback`)

Valid for: Image. Cell property of Grid object

Grid object

Each cell element is an integer scalar.

Determines the background color for the cell (when it is not selected). The value is a color expressed as a numeric scalar ($256 \times \text{Blue} + \text{Green} + \text{Red}$). The special value of `-1` means use the Grid background color for this cell; this is the default.

Image object

The `colorback` property represents a single color, specified either as a numeric scalar or a three-element vector of Red Green Blue values, each in the range 0 to 255. It specifies the background color used to fill blank areas created when an image is rotated, sheared, or extended.

The 'colorgrid' property

(Alternative name: `colourgrid`)

Integer Scalar or Length-3 Vector

Valid for: Chart

This property allow you to set a specific color for the grid lines (if any) of the chart. It is specified as a single color value. It can be expressed as a vector of three separate RGB values each in the range 0 - 255, or as a single integer which encodes the three values in base 256 as $256 \times \text{Blue} + \text{Green} + \text{Red}$. The default is the special value `-1`, meaning use the Chart object's foreground color.

When you read back this property, APLX always returns the single-integer encoded form.

This property is ignored if the Chart's `monochrome` property is set to 1.

The 'colorhead' property

(Alternative name: `colourhead`)

Integer Scalar or 3-element Integer Vector

Valid for: Grid

The `colorhead` property for a Grid can be set as either a single RGB encoded integer ($256 \times \text{Blue} + \text{Green} + \text{Red}$) or as three Red Green Blue values, each 0 to 255. It determines the background color used for header cells. The special value `-1` means use the Grid control's foreground color. The special value `-2` means use the default (light gray).

The 'colorlegend' property

(Alternative name: `colourlegend`)

Integer Scalar or Length-3 Vector

Valid for: Chart

This property allow you to set a specific color for the text used to display the legend (if any) of the chart. It is specified as a single color value. It can be expressed as a vector of three separate RGB values each in the range 0 - 255, or as a single integer which encodes the three values in base 256 as $256 \times \text{Blue} + \text{Green} + \text{Red}$. The default is the special value `-1`, meaning use the Chart object's foreground color.

When you read back this property, APLX always returns the single-integer encoded form.

This property is ignored if the Chart's `monochrome` property is set to 1.

The 'colormarker' property

(Alternative name: `colourmarker`)

Integer Vector or Scalar (*see description*)

Valid for: Series

Usually, any marker associated with a Series object is drawn in the color set for the series as a whole (i.e. the `color` property, which is either chosen automatically or set explicitly). If you wish, you can display the marker in a different color by setting the `colormarker` property. You can also specify a background color for filling the marker. The property is specified in the same way as the ordinary `color` property. It can be either as a single color value (vector of three separate RGB values, or a single integer encoding the three values), in which case it is used for both the foreground and background colors, or as two color values (length six vector of RGB values, or two integers encoding the color values), in which case the foreground color is used to outline the marker, and the background color is used to fill it if the `fillmarker` property is 1. The default values of `-2 -2` mean use the same color as is used for the series as a whole.

The `colormarker` property is ignored if the Chart's `monochrome` property is set to 1.

The 'colornote' property

(Alternative name: colournote)

Integer Scalar or Length-3 Vector

Valid for: Chart

This property allow you to set a specific color for the text used to display the 'note' (if any) of the chart. It is specified as a single color value. It can be expressed as a vector of three separate RGB values each in the range 0 - 255, or as a single integer which encodes the three values in base 256 as $256 \times \text{Blue} + \text{Green} + \text{Red}$. The default is the special value -1 , meaning use the Chart object's foreground color.

When you read back this property, APLX always returns the single-integer encoded form.

This property is ignored if the Chart's `monochrome` property is set to 1.

The 'colortext' property

(Alternative name: colourtext)

Cell property of Grid object

Each cell element is an integer scalar

Determines the color of text in the cell. The value is a color expressed as a numeric scalar ($256 \times \text{Blue} + \text{Green} + \text{Red}$). The special value of -1 means use the Grid foreground color for this cell; this is the default.

The 'colortitle' property

(Alternative name: `colourtitle`)

Integer Scalar or Length-3 Vector

Valid for: Chart

This property allow you to set a specific color for the text used to display the title (if any) of the chart. It is specified as a single color value. It can be expressed as a vector of three separate RGB values each in the range 0 - 255, or as a single integer which encodes the three values in base 256 as $256 \cdot \text{Blue} + \text{Green} + \text{Red}$. The default is the special value `-1`, meaning use the Chart object's foreground color.

When you read back this property, APLX always returns the single-integer encoded form.

This property is ignored if the Chart's `monochrome` property is set to 1.

The 'colour' property

Synonym for the `color` property

The 'cols' property

Integer Scalar

Valid for: Grid

The number of data (scrolling) columns in the Grid control. You typically set this property when you create the Grid to fit the size of the data you want to display.

The 'colsize' property

Valid for: Grid (*Special syntax applies*)

You can set the size of the columns of a Grid control using the syntax:

```
WindowRef.ControlName.colsize Cols Sizes
```

or:

```
'WindowName.ControlName' □WI 'colsize' Cols Sizes
```

where `Cols` is a scalar or vector list of column numbers, and `Sizes` is a matching vector (or scalar) of the column sizes, in pixels. A column number of 0 sets the default size for all columns which are not explicitly set. *Note: At least one of the `Cols` and `Sizes` parameters must be a vector, otherwise the statement will be interpreted as an attempt to read back the current sizes.*

You can read back the current sizes using the syntax:

```
R ← WindowRef.ControlName.colsize Cols
```

or:

```
R ← 'WindowName.ControlName' □WI 'colsize' Cols
```

Note that, depending on the flags you have set in the `style` property, the user may be able to resize the columns.

The 'contents' property

Character vector

Valid for: Browser, OLEContainer. System

System object

For the System object, the `contents` property represents the contents of the clipboard as an APL array. For example, you can copy a range of cells from Excel to the clipboard, and read it into APL as a numeric or nested array. The clipboard formats which are accepted when you read this property are (in order of preference):

1. The APLX binary format, which supports any arbitrary array
2. SLK (Spreadsheet format), two-dimensional arrays only
3. CSV (Comma-Separated Variables), two-dimensional arrays only
4. TSV (Tab-Separated Variables), two-dimensional arrays only

If none of these formats is found, a text vector is returned.

If you write an array to the `contents` property, APLX will make the data available on the clipboard in the following formats:

1. The APLX binary format, which supports any arbitrary array but is supported by APLX only
2. HTML, as a table (two-dimensional arrays only)
3. Unicode text (monadic format of the array)

Browser

For a Browser object, the `contents` property is read-only. It returns a character vector (with embedded carriage returns), containing the raw HTML text of the page currently displayed in the control.

OLEContainer (*Windows only*)

When you read the `contents` property for an OLEContainer control in which a document is embedded or linked, APLX returns a character vector containing the internal representation of the document (or the link) in the container. It is encoded as a series of bytes, and includes information about the type of the document as well as its contents.

You should never change the contents of this vector, but you can store it in a file and write the same vector back to an OLEContainer control to cause it to re-display the same document. This has the effect of starting up the OLE server application associated with the document.

If there is no document loaded, reading `contents` returns an empty vector.

If you write an empty vector to `contents`, a dialog is displayed inviting the user to select a document to embed or link in the OLEContainer.

The 'conversionerrorvalue' property

Numeric scalar

Valid for: Grid

The value to return in the `value` cell property if a numeric cell cannot be converted to a valid number. The default is a large negative number.

The 'cookie' property

Read-only, Character Vector

Valid for: HTTPClient

Once you have used the `Get` method of an `HTTPClient` object to retrieve the a page from a web-site, the `cookie` property contains the text of any cookie sent by the web server.

The 'copies' property

Integer scalar in range 1 to N

Valid for: Printer

The `copies` property contains the number of copies which will be printed in the current print job. Usually, you do not need to set this explicitly, because the user selects the number of copies in the print-job dialog. You can read this property to determine how many copies the user has requested, and you can also set this property before calling the `Job` method to pre-set a particular number of copies. For unattended printing, you can set this property before calling the `Open` method.

The 'count' property

Read-only, Integer scalar (for `GetMail`), Two-element integer vector (for `Tree`)

Valid for: `GetMail`, `Tree`

For a `Getmail` object, the `count` property contains the number of e-mails available on the POP3 server. It is valid only when you are successfully connected to the server.

For a `Tree` object, the `count` property returns a two-element vector. The first element is the total number of nodes in the tree, and the second is the number of nodes that are fully visible in the control. Under Linux, the second element is not available and will always be returned as 0.

The 'custom' property

Implemented under Windows only

Integer vector of up to 16 elements

Valid for: ChooseColor

The pre-defined ChooseColor dialog can optionally allow the user to choose from and define 'custom' colors, in addition to the default colors which appear in the dialog. (Custom colors are allowed by default, unless you specify 2 for the `style` property). Setting the `custom` property before the dialog is displayed allows you to specify the custom colors which will appear, and reading the `custom` property after the dialog has been closed by the user allows you to retrieve the values of the custom colors which the user may have defined or changed.

The colors in the custom property are each encoded in base 256 as $256 \times \text{Blue} + \text{Green} + \text{Red}$, where the Blue, Green and Red values are in the range 0 to 255.

The 'data' property

Any APL array or overlay

Valid for: All objects

The `data` property allows you to associate arbitrary APL data with any object. The APL data can be retrieved later, for example during a callback function. The data is stored outside the workspace, in system memory.

Using the data property can help make your applications better structured, allowing information associated with an object to be stored with it rather than held in global variables in the workspace.

If you reference the `data` property before assigning to it, APLX reports a VALUE ERROR.

You can also assign arbitrary APL data to 'delta' properties, allowing you to have as many distinct variables associated with an object as you like.

Special considerations for Client-Server versions of APLX

Because the whole System Class sub-system runs on the Client machine, the data contained in the `data` property is held (in memory) on the Client machine, as a variable in APLX 32-bit format (even if

the APL session which wrote the variable is a 64-bit implementation of APLX). The data must therefore be representable as a 32-bit object, which means it must be smaller than 2GB and must not contain more than 2147483647 elements. An array which contains 64-bit integer numbers of magnitude bigger than $2 \cdot 31$ will be converted to float data, and precision may be lost.

The 'date' property

Read-only, Character Vector

Valid for: GetMail

The `date` property contains the timestamp of an e-mail, as a character vector. It is a read-only property, and is valid only after you have called the `GetMessage` method to retrieve the e-mail from the POP3 server.

The date is normally formatted in the form 'Tue, 16 Nov 2004 12:17:40 +0000', where the last part is the offset from Greenwich Mean Time as HHMM.

The 'def' property

Not currently implemented

The 'default' property

Character vector (or integer scalar for the `MsgBox` class)

Valid for: `OpenFile`, `SaveFile`, `MsgBox`

Under Windows and Linux, for the `SaveFile` and `OpenFile` pre-defined dialogs, the `default` property determines the default file extension which will be added if the user specifies a file name without a file extension, for example 'txt'. File extensions must not exceed three characters, and the leading period should be omitted. It is ignored under MacOS.

For the `MsgBox` pre-defined dialog, the `default` property is an integer in the range 1 to 4, representing the default button which will be selected if the user presses the Enter key.

The 'deleteonread' property

Boolean Scalar

Valid for: GetMail

The `deleteonread` property determines whether or not e-mail messages are automatically marked for deletion on the POP3 server once they have been read. If this property is 1, they will be deleted after you have successfully retrieved them using the `GetMessage` method. If it is 0, they will be left on the server.

The messages are not actually deleted until the POP3 session has been successfully completed and closed using the `Close` method. If an error occurs, or if the session is aborted because the connection is lost, they will not be deleted.

The 'directory' property

Character vector

Valid for: ChooseDir, System

ChooseDir object

The ChooseDir pre-defined dialog allows the user to select a directory on disk. Before the dialog is displayed, you can optionally set the `directory` property to specify the initial directory which will be displayed in the dialog. After the user has made the selection, read the `directory` property to retrieve the full path name of the selected directory.

System object

The `directory` property of the System object returns the directory from which the running program (either APLX itself, or your packaged application) was loaded. It is a read-only property.

The 'docstate' property

Read-only, Integer scalar

Valid for: OLEContainer (*Implemented under Windows only*)

The `docstate` property returns information about the state of the document in an OLEContainer. The value returned is one of:

- 0 = There is no OLE object in the container.
- 1 = There is an OLE object in the container, but its server application isn't currently running.
- 2 = The OLE object's server is running.
- 3 = The OLE object is open in a separate window.
- 4 = The OLE object is activated inplace, but hasn't yet merged its menus or toolbars.
- 5 = The OLE object is activated inplace and menus and toolbars have been merged

The 'doublebuffered' property

Boolean scalar

Valid for: All visible controls and windows

The `doublebuffered` property is a boolean scalar which, under Windows, determines whether drawing takes place to an off-screen bitmap before the window is updated. The default is false (0). Setting `doublebuffered` to true (1) takes more memory, but may reduce screen flicker especially if you are moving shapes on screen.

Under MacOS and Linux, this property has no effect, since the operating system determines whether an off-screen bitmap is used for drawing. Double-buffered drawing is always used under MacOS X and later.

The 'dragsource' property

Valid for: All visible controls

Read-only, character vector

The `dragsource` property is used in drag-and-drop operations. When a drag-and-drop callback occurs (i.e. one of `onDragStart`, `onDragEnd`, `onDragOver`, `onDragEnter`, `onDragLeave`, `onDragDrop`), the `dragsource` property contains the name of the control which is being dragged.

The 'droptarget' property

Valid for: All visible controls

Read-only, character vector

The `droptarget` property is used in drag-and-drop operations. When an `onEndDrag` callback is invoked, the `droptarget` property contains the name of the control on to which the source control is being dragged.

The 'enabled' property

Boolean scalar

Valid for: Series, Timer, and all visible objects

The `enabled` property determines whether an object is active and capable of responding to events such as mouse down. If `enabled` is 1 (the default), the control is active and potentially responds to events.

Text objects (such as static text, button, check box, radio button, list, edit, menus) display their text as greyed out when they are disabled.

For a Timer object, the `enabled` property determines whether or not the timer will generate `onTimer` events at the interval set by the `interval` property.

For a Series object, the `enabled` property determines whether the Series is active. If it is 0, the Series does not appear, and is disregarded when the parent Chart computes the scale of the chart.

The 'eventmask' property

Not currently implemented in APLX.

The 'events' property

Read-only, Nested vector of character vectors

Valid for: All objects

The `events` property returns a nested vector of the names of the events which are meaningful for a particular object (for example, 'onClick'). These are the event properties which can be triggered as a result of user action or by the system, and for which you might want to define APL callbacks.

Under Windows, for an OCX/ActiveX control, OLEContainer, or OLE Server Application, events defined by the external software will be included in the list, preceded by 'onX'.

Note: You can also define callbacks for events which are not in this list (for example, you could define an 'onClick' callback for a Timer object). This is valid, but such a callback will never be triggered unless you explicitly call it using the `Trigger` method.

The 'extent' property

Two-element numeric vector

Valid for: All displayable objects

The `extent` property determines the height and width of an object, in the `scale` of its parent. The first element of the `extent` property is the height, and the second is the width of the object. (This is exactly the same as the `size` property, except that the `scale` used for the `size` property is that of the object itself, not its parent.)

The 'family' property

Integer Scalar

Valid for: Socket

The `family` property specifies the address family of the socket. It can be one of the following values:

<i>Value</i>	<i>C Name</i>	<i>Description</i>
1	AF_UNIX	local to host (pipes, portals)
2	AF_INET	internetwork: UDP, TCP, etc.
3	AF_IMPLINK	arpanet imp addresses
4	AF_PUP	pup protocols: e.g. BSP
5	AF_CHAOS	mit CHAOS protocols
6	AF_NS	IPX or XEROX NS protocols
7	AF_ISO	ISO protocols
8	AF_ECMA	european computer manufacturers
9	AF_DATAKIT	datakit protocols
10	AF_CCITT	CCITT protocols, X.25 etc
11	AF_SNA	IBM SNA
12	AF_DECnet	DECnet
13	AF_DLI	Direct data link interface
14	AF_LAT	LAT
15	AF_HYLINK	NSC Hyperchannel
16	AF_APPLETALK	AppleTalk
17	AF_NETBIOS	NetBios-style addresses
18	AF_VOICEVIEW	VoiceView
19	AF_FIREFOX	Protocols from Firefox
21	AF_BAN	Banyan
22	AF_ATM	Native ATM Services
23	AF_INET6	Internetwork Version 6
24	AF_CLUSTER	Microsoft Wolfpack
25	AF_12844	IEEE 1284.4 WG AF
26	AF_IRDA	IrDA
28	AF_NETDES	Network Designers OSI & gateway enabled protocols

The default is 2, the value required for Internet protocols.

The 'file' property

Character vector (or a nested vector of character vectors)

Valid for: Movie, OpenFile, SaveFile, Image, Picture, OLEContainer, System

The `file` property allows you to set or retrieve a file name. The exact meaning depends on the object:

OpenFile and SaveFile pre-defined dialogs

For the two pre-defined dialogs which allow the user to select a file name, the `file` property contains the filename. If you set this property to a character vector before displaying the dialog, it determines the initial choice which will be presented in the dialog. You can set either:

- A full path name
- Just the file part, in which case you the current working directory is assumed.
- Just a directory (ending in a directory-separator character), in which case there is no initial file selection in the dialog, but the initial directory will be set.

After the dialog has been closed by the user, the `file` property contains the fully-qualified name of the selected file, usually as a character vector. In the case of the OpenFile dialog, you can optionally allow multiple files to be selected, by setting the style value 1. In this case, the `file` property will return a nested vector of file names.

Picture object

For a Picture object, the `file` property determines the image which will be displayed in the picture. Under Windows, it should be the name of a Windows bitmap, icon, or JPEG image file (.bmp, .ico, .jpg or .jpeg). Under MacOS, it should be the name of a JPEG or PICT file, or any image type supported by Apple's QuickTime software. Under Linux, the supported types depend on the operating system version but will typically include .png .xpm .jpg .jpeg .ico and .bmp files.

If you write an empty vector to the `file` property, a dialog is displayed allowing the user to choose an image file.

Image object

For an Image object, the `file` property contains the name of the file (if any) from which the image was loaded. Writing to the property causes the named file to be loaded. If you write an empty vector to the `file` property, a dialog is displayed allowing the user to choose an image file, which can be of any format supported by ImageMagick.

Movie object

When you initially create a Movie object, a blank rectangle is displayed because the system does not know what movie you wish to play. You select the particular movie you want by setting the `file` property for the movie object. (Note that you must do this after setting the style property). There are two ways you can specify the file:

(a) If you set the `file` property to be a file name, that is used as the movie file. The name can either be a simple file name (in which case the default volume and directory is assumed), or a full pathname. For example:

```
MyWin.Mov.file←'Mydisk:Movies:Advert'
```

(b) If you set the file property to be an empty vector, APLX displays the standard file selection dialog box, allowing the user to choose a movie file. This is then used for the movie object automatically.

A special feature of movie objects is that setting the `file` property has the side effect of re-sizing the object to the default size for the selected movie. The movie will usually display best if the size of the object is the same as (or an integral multiple of) the original movie size. You can change the size of the movie object if you wish, but this must be done after setting the `file` property.

OLEContainer object

For an OLEContainer control, the `file` property determines the name of the document. If the OLE container is linked to a document, reading the `file` property allows you to find out its name.

Writing to the `file` property causes the container to attempt to load the document (or link to it, if the `style` is 1). If you write an empty vector, a dialog is displayed allowing the user to choose a document.

System object

If you have packaged your APL workspace as a standalone application, it may be launched by the user dragging one or more documents on to the application icon or double-clicking a document (for this to work, you need to have set up the association between the file type/extension and the application). Under Windows, the user may also have started the application from the command-line. The System object's `file` property allows your APL program to respond correctly to these user actions. It is a read-only property.

Under MacOS, `file` contains the full names of the document or documents which you should open or print. If there are several file names, they will be delimited by carriage-return characters. The `action` property indicates whether you should open or print them. The `onOpen` event is triggered if your application is already running.

Under Windows and Linux, there are two possibilities. If the user dragged a document icon on to your application icon (or double-clicked the document icon), `file` will contain the full filename of the document. If the user started your application from the command-line, `file` will contain the command-line options which were entered.

The 'filled' property

0 or 1

Valid for: Line, Rectangle, RoundRect, Arc

For a graphic object, the filled property indicates whether the object is filled with the foreground color (1), or not (0). The default is 0. This property can be set for a Line, but is ignored. For an arc which does not describe a full 360 degrees, the effect is to display a filled wedge (such as you would have in a 'pie' chart).

The 'fillmarker' property

Boolean Scalar

Valid for: Series

Where markers are drawn for the points of a Series displayed on a Chart object, by default they are drawn in outline only. If you set the `fillmarker` property of the Series object to 1, the markers will be filled with the current marker background color (as set by the `colormarker` property), provided it is one of the following shapes: `o` `φ` `e` `▽` `△` `◇` `□`

See also the `marker` and `colormarker` properties.

The 'fillpattern' property

Integer Scalar

Valid for: Series

Normally, bars and pie slices associated with a Series object are filled with a solid block of color. If you wish, you can specify a different fill pattern for the series, by assigning one of the following values to the `fillpattern` property:

1=Solid, 2=Horizontal grid pattern, 3=Vertical grid pattern, 4=Forward diagonal, 5=Backward diagonal, 6=Cross pattern, 7=Diagonal cross pattern

Note: If the Chart's `monochrome` property is set to 1, and you haven't specified your own fill pattern, a default pattern from one of the above will automatically be assigned to the Series object to distinguish it from other series on the chart.

The 'filter' property

Character vector

Valid for: `OpenFile`, `SaveFile`

The `filter` property sets one or more *filters* (file extensions or types which will be allowed), for use in the `OpenFile` and `SaveFile` pre-defined dialogs.

It comprises one or more sets of descriptive text, a | character, and a semi-colon delimited list of supported filters. Multiple filters should be separated by a vertical bar.

Under Windows and Linux, you specify file extensions as file wildcards. For example, the filter `'Text files|*.txt;*.log|All files|*.*'` gives the user the choice of selecting "Text files" (in which case extensions `.txt` and `.log` are accepted), or "All files" (any extension is accepted).

Under MacOS, you can choose to filter either using file extensions (in which case the syntax is identical to that used for Windows and Linux), or using file types. In the latter case the filters are four-character file types rather than file extensions beginning with the wildcard `*.*`. For example: `'Picture files|JPEG;PICT'` would display the description "Picture files" to the user, and allow selection of files of type JPEG and PICT. The file type `'*****'` matches all file types. You can also combine both types of filter; for example, `'Text files|*.txt;TEXT'` would match files which either have the extension `.txt` or which are of type TEXT.

The 'filterindex' property

Integer scalar

Valid for: `OpenFile`, `SaveFile`

The `filterindex` property determines which of the file extensions or types (of the list set using the `filter` property) is displayed in an `OpenFile` or `SaveFile` pre-defined dialog. The value is the index (in index origin 1) of the filter to display.

Usually you set `filterindex` before displaying the dialog. You can also read it when the dialog ends, to see whether the user has changed it. This is useful to decide the file type to use. However, under

Windows and Linux, you should also check the file extension (if any), because the user can type a full filename and extension without changing the selected filter.

The 'firstvisible' property

Integer scalar or two-element vector

Valid for: List, Grid

For a List object, the `firstvisible` property is an integer scalar which represents the first visible item at the top of the list box. Thus, if the list box has been scrolled down three lines, it will have the value 4. You can write to this property to force a scroll so that a specific item is at the top of the visible portion of the list; however, because you the list will not scroll beyond the last item, you should read back the property if you need to know exactly which item ended up as the first visible item.

For a Grid object, `firstvisible` is a two-element integer vector. It allows you to read or set the top-left position of the visible portion of the Grid. It is the same as the first two elements of the `view` property.

The 'font' property

A character vector or nested vector - see below

Valid for: Document, Static, Label, Grid, Edit, RichEdit, Button, ToolButton, Radio, Check, List, Combo, Page, Selector, Frame, ChooseFont (*Windows and Linux only*), Printer, Chart

The font in which text objects display their text is set by the `font` property. In general, this is a nested vector of four elements, but you do not have to supply all four, and if you supply a simple character vector it will be taken as the font name. The four elements are:

1. Font name as a character vector. If you specify an empty vector, the font is unchanged. Otherwise you can specify the name of the font you want to use. If you specify a font which is not installed, or which is not compatible with the other characteristics you specify, the system will choose an alternative font for you. To select the APL font, you can specify either 'APLX Mac' or 'APLX Upright'.
2. Font size, in the current units of the object (see the `scale` property).
3. Font style as an integer scalar, from one of the following:

0	Plain
1	Bold
2	Italic
4	Underlined
8	Hollow (supported under MacOS only)
16	Strikeout (not supported under MacOS)

You can also specify the sum of these, for example 3 for Bold and Italic.

4. The name of the font character set, as a character vector. On the Macintosh, this is always 'mac'. Under Windows or Linux, this is one of 'ansi' 'default', 'symbol' 'mac' 'greek' 'turkish' 'baltic' 'easturope' and 'oem'. The character set is used to select a version of the font capable of displaying characters appropriate to the chosen locale.

For example, if the `scale` property is currently set to 3 (points), and you want a particular object to display in 16-point Helvetica italic, you might enter:

```
Win19.Subtitle.font←'Helvetica' 16 2
```

Note: Remember that the default scale is in character units, so that if you have not changed the `scale` property the above example would give a font 16 times the default size, or 256 points.

The default font depends on the system and the user's default settings.

For a Chart object, the Size parameter of the `font` property is ignored. The property is used to determine the base font of the Chart, which by default is used to draw each text element of the Chart at a size which depends on the window size.

Under Windows or Linux, you can ask the user to select a font using the ChooseFont pre-defined dialog. Before showing the dialog using the `Show` method, you can set the `font` property to choose the initial selection which will be displayed when it appears. When the user has made the selection, read back the `font` property to retrieve the selected font and style.

The 'fontaxis' property

Nested Vector

Valid for: Chart

The `fontaxis` property allow you to specify the font for tick labels and axis labels of a Chart object. Any changes you make to this property overrides the setting of the base `font` property of the object.

The font details are specified in a similar way to ordinary `font` properties, as a nested vector of (Font name) (Size) (Style) (Character set). However, the Size is defined in a special way. If it is set to a value between 0 and 1, it means that the height of the font should be the specified proportion of the

window size, defined as the smaller of the window height and width. Values in the region of 0.03 to 0.05 are reasonable. This is recommended for resizable charts, to ensure that the labels remain in proportion to the chart as a whole. (If the font gets too small, a lower limit is imposed). If the font size is specified as an integer greater than 1, it is interpreted as a font size in Points, and will be fixed irrespective of the size of the Chart. This is likely to mean that the chart does not display well at very small or large sizes.

If the Font name is an empty vector, the base font family is used. If the Size or Style element is `-1`, the corresponding font attribute is left unchanged.

The 'fontlegend' property

Nested Vector

Valid for: Chart

The `fontlegend` property allow you to specify the font for the Legend of a Chart object. Any changes you make to this property overrides the setting of the base `font` property of the object.

See the description of the `fontaxis` property for details of how it is specified.

The 'fontnote' property

Nested Vector

Valid for: Chart

The `fontnote` property allow you to specify the font for the `note` of a Chart object. Any changes you make to this property overrides the setting of the base `font` property of the object.

See the description of the `fontaxis` property for details of how it is specified.

The 'fonts' property

Read-only, nested vector of character vectors

Valid for: System, Printer (*Windows only*)

The read-only property `fonts` applies to the System object. It returns a nested vector containing the names of the screen fonts installed on the system:

```

      '#' ⍵I 'fonts'
Abadi MT Condensed
Abadi MT Condensed Extra Bold
Abadi MT Condensed Light
Algerian
American Uncial
Andy
APL385 Unicode
APLX Upright
Arial
..etc

```

(Note that the fact that a font is installed does not guarantee that it is available in a particular size, style and character set).

Under Windows only, the `fonts` property also applies to the Printer object. It returns a nested vector of the names of the fonts installed on the currently-selected printer, together with the TrueType fonts which can be downloaded to the printer.

The 'fontstyle' property

Cell property of Grid object

Each cell element is an integer scalar

Determines the style of the text within the cell. The value is the sum of:

0	Plain
1	Bold
2	Italic
4	Underlined
8	Hollow (supported under MacOS only)
16	Strikeout (not supported under MacOS)

The special value of -1 means use the default font style of the Grid.

The 'fonttitle' property

Nested Vector

Valid for: Chart

The `fontnote` property allow you to specify the font for the Title of a Chart object. Any changes you make to this property overrides the setting of the base `font` property of the object.

See the description of the `fontaxis` property for details of how it is specified.

The 'format' property

Valid for: Image, Grid

Image object

Character Vector

For an Image object, the `format` property is a text string which indicates the graphics format used to store the image on disk. After you have called the `Load` method to read an image file, the `format` property will reflect the original format of the file, for example 'JPEG' or 'GIF'.

If you want to save the image in a different format, you can write a different string to the `format` property before calling the `Save` method.

The list of supported formats can be read from the `formats` property. However, some of these may be valid for reading only - see the ImageMagick documentation for more details.

Grid object

For the Grid object, the `format` is a Cell property, where each cell element is a character vector. It determines the format used to display numeric values in cells of a Grid object. The format string is a character vector of up to three parts, where each part is separated by a semicolon. If you supply just one part, it applies to all numbers. If you supply two parts, the first applies to positive numbers or zero, and the second to negative numbers. If you supply all three parts, the first applies to numbers greater than zero, the second to less than zero, and the third to zero.

Each part comprises a set of characters giving the format for the number. These comprise:

Character Meaning

- 0** Required digit. If the number has a digit in the position where the '0' appears in the format, then that digit is displayed, otherwise 0 is displayed. Thus, the positions of the leftmost '0' before the decimal point and the rightmost '0' after the decimal point in the format determine the range of digits that are always displayed.
- #** Optional digit. If the number has a digit in the position where the '#' appears in the format, then that digit is displayed. Otherwise, nothing is displayed in that position.
- .** Decimal point. The first '.' character in the format string determines the location of the decimal separator in the display. (The actual character used as a the decimal separator may depend on the locale set in the operating system).
- ,** Thousands separator. If the format string contains one or more ',' characters, the output will have thousand separators inserted between each group of three digits to the left of the decimal point. The placement and number of ',' characters in the format string does not affect the output, except to indicate that thousand separators are wanted. (The actual character used may depend on the locale set in the operating system.)
- E+** Scientific notation. If any of the strings 'E+', 'E-', 'e+', or 'e-' are contained in the format string, the number is formatted using scientific notation. A group of up to four '0' characters can immediately follow the 'E+', 'E-', 'e+', or 'e-' to determine the minimum number of digits in the exponent. The 'E+' and 'e+' formats cause a plus sign to be output for positive exponents and a minus sign to be output for negative exponents. The 'E-' and 'e-' formats output a sign character only for negative exponents.
- ' "** Quotes. Characters enclosed in single or double quotes are output unchanged.

Setting this value automatically sets the cell type to Numeric.

The 'formats' property

Read-only, Nested Vector of Character Vectors

Valid for: Image

The `formats` property of an Image object returns a list of all the formats supported by the underlying ImageMagick library, as a nested vector of text strings. For example:

```
Image1←'[]' NEW 'Image'
Image1.formats
A ART AVI AVS B BIE BMP BMP2 BMP3 C CACHE CAPTION CIN CIP CLIP CLIPBOARD CMYK C
MYKA CUR CUT DCM DCX DOT DPS DPX EMF EPDF EPI EPS EPS2 EPS3 EPSF EPSI EPT
EPT2 EPT3 FAX FITS FPX FRACTAL G G3 GIF GIF87 GRADIENT GRAY HDF HISTOGRAM
```

HTM HTML ICB ICO ICON JBG JBIG JNG JP2 JPC JPEG JPG K LABEL M M2V MAP MAT
MATTE MIFF MNG MONO MPC MPEG MPG MSL MTV MVG NULL O OTB P7 PAL PALM PATT
N PBM PCD PCDS PCL PCT PCX PDB PDF PFA PFB PGM PGX PICON PICT PIX PJPEG PL
ASMA PNG PNG24 PNG32 PNG8 PNM PPM PREVIEW PS PS2 PS3 PSD PTIF PWP R RAS RG
B RGBA RLA RLE SCR SCT SFW SGI SHTML STEGANO SUN SVG SVGZ TEXT TGA TIF TIF
F TILE TIM TTC TTF TXT UIL UYVY VDA VICAR VID VIFF VST WBMP WMF WMFWIN32 W
MZ WPG X XBM XC XCF XPM XV XWD Y YCbCr YCbCrA YUV

See the `format` property for more information.

The 'from' property

Character Vector

Valid for: `GetMail`, `SendMail`

The `from` property contains the e-mail address of the sender of an e-mail as a character vector.

For the `SendMail` object, you should set this property before calling the `SendMessage` method.

For the `GetMail` object, this property is read-only, and is valid only after you have called the `GetMessage` or `GetSummary` method to retrieve an e-mail (or summary of an e-mail) from the POP3 server.

The 'gridlines' property

Boolean scalar

Valid for: `Grid`

Determines whether the grid lines within the data area are visible. (Default 1)

The 'gridwidth' property

Integer Scalar

Valid for: Chart

The `gridwidth` property allows you to specify the line width (in pixels) for the Chart object's grid lines, as an integer scalar. The default is `-1`, which means use the value set in the Chart object's `linewidth` property. (Note that the gridlines will appear only if the appropriate values in the `style` property are set.)

The 'group' property

An integer in the range 0 to 255

Valid for: Menu, Radio, ToolButton

Menu items

The `group` property is used to make Menu items behave like radio buttons. It indicates which group of mutually-exclusive menu buttons the object belongs to. When one item in the group is selected (by the user or under program control), all others in the same group on the same menu are de-selected. The `group` property is an arbitrary number in the range 0 to 255 which serves to identify the group within the window. If the value is 0, the control does not exhibit radio-button behavior.

Radio buttons

The `group` property of a Radio button determines which group (within a particular control) the button belongs to. Only one Radio button in the group can be selected at any time, so that if one is selected, all others in the same group are automatically de-selected.

Usually, this grouping occurs automatically (typically, you place Radio buttons in a Frame object). The initial `group` property for the first Radio button to be created as the child of a Frame or Window is 0. If you create further Radio buttons one after another with the same parent, the `group` property for each new button is the same as that of the previous button, so each button in the series belongs to the same group.

If you create a *different* control or controls in between, the `group` property is incremented by one for the next series, so that the second series of Radio buttons behaves independently of the first.

Alternatively, you can override this behavior and set the Radio button's `group` explicitly, as an integer scalar in the range 0 to 255.

ToolButtons

For a `ToolButton`, the `group` property determines which group of similar buttons the control belongs to, and thus whether it behaves like a radio button. If it is 0, the button behaves like an Action button. See the description of the `ToolButton` for details.

The 'handle' property

Read-Only (*except for Socket*), Integer Scalar

Valid for: Check, Radio, Button, Progress, Trackbar, Spinner, Scroll, List, Combo, Edit, RichEdit, Movie, Splitter, Selector, Page, Tree, Frame, Printer, Image, ImageList, Socket

The `handle` property is generally used for low-level programming only. Its value is the internal value which identifies the underlying control to the operating system.

It can be passed to external calls which require a handle argument via `□NA` or Auxiliary Processors. If no valid handle is available (for example, if the object is not open), 0 is returned.

For a `Socket` object, the `handle` property is used in a special way for server sockets accepting requests for connection from clients. See the documentation on the `Socket` object for details.

The 'hdc' property

Synonym for the `winptr` property

The 'headcols' property

Integer scalar

Valid for: Grid

The number of heading (fixed) columns (Default 1). You typically set this property when you create the grid (before setting the `cols` property)

The 'header' property

Read-only, Character Vector

Valid for: GetMail

The `header` property returns the raw header of an e-mail. It is valid only after you have called the `GetMessage` (or `GetSummary`) method to retrieve an e-mail from the POP3 server. It is formatted as a character vector, with embedded carriage return (␣) characters separating lines. You can conveniently use `BOX` to convert this to a matrix.

An e-mail header comprises a keyword, ending in a colon, followed by a text string which is the value associated with the keyword. This can continue on successive lines. For example:

```
MailObj.GetMessage 1
MailObj.header
Return-path: <bounce_204417_131880@b2.mx0.net>
Received: from pop3.isp.net by mailstore
  for microapl@microapl.co.uk id 1Cp0oK-00055Z-BL;
  Fri, 14 Jan 2005 10:36:08 +0000
Message-Id: <d1to1.204417.131880@m2.mx0.net>
Date: Fri, 14 Jan 2005 10:31:00 +0000
From: "VSJ" <VSJ-204417-131880z@m2.mx0.net>
To: microapl@microapl.co.uk
Subject: VSJ Developer Newswire - 14th Jan 2005
List-Owner: <mailto:abuse@mx0.net>
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7Bit
```

APLX automatically extracts the most important of these header fields into specific properties of the `GetMail` object such as `subject`, `date`, `to` and `from`. However, sometimes you may want to examine the full header to extract more information, such as the path by which the e-mail reached your mailbox (held in a series of one or more 'Received:' fields).

The 'headrows' property

Integer scalar

Valid for: Grid

The number of heading (fixed) rows (Default 1). You typically set this property when you create the grid (before setting the `rows` property)

The 'highlightbold' property

Integer vector

Valid for: Tree

The `highlightbold` property is an integer vector which contains a list of the IDs of all the Tree-object nodes which are shown in bold face.

The 'highlightcut' property

Integer vector

Valid for: Tree

Under Windows and Linux, the `highlightcut` property is an integer vector which contains a list of the IDs of all the Tree-object nodes which are highlighted for cutting. It is ignored under MacOS.

The 'highlightdrop' property

Integer vector

Valid for: Tree

Under Windows and Linux, the `highlightcut` property is an integer vector which contains a list of the IDs of all the Tree-object nodes which are highlighted for dropping. It is ignored under MacOS.

The 'highlightselect' property

Integer vector

Valid for: Tree

The `highlightselect` property is an integer vector which contains a list of the IDs of all the Tree-object nodes which are highlighted as if selected.

The 'highvalues' property

Numeric Vector

Valid for: Series

The `highvalues` property sets the data used for the high values of a Series object displayed on a High-Low-Open-Close or Candlestick chart. It is a numeric vector.

You should normally provide the same number of values for each of the `openvalues`, `closevalues`, `highvalues` and `lowvalues` properties. However, the `openvalues` and `closevalues` properties are optional, and can be left as an empty vector.

The 'host' property

Character Vector

Valid for: GetMail, SendMail, APL (Child Task), System object

GetMail and SendMail

Before you can use the GetMail or SendMail objects, you need to specify the internet address of the host you are using. For the GetMail object, this is the address of your POP3 server. For the SendMail object, it is the address of your SMTP server. You will probably also need a user name and password. The values to use will normally be supplied by your Internet Service Provider (ISP) or network administrator.

The host address can be specified either as a domain name (such as 'pop3.myisp.com'), or directly as an internet node address (such as '168.9.211.53').

For example:

```
M←' ' NEW 'GetMail'
M.host←'pop3.supernet.com'
M.user←'microapl'
M.password←'secret'
M.Open
```

0

APL (Child Task)

If you are running a Client-Server implementation of APLX, you can use the `host` property of the APL object to specify on which machine the new APL task should run. There are three possibilities:

- You can set the `host` property to an empty string. In this case, the new APL task will be created as a local 32-bit session, running under the same process ID as the Client (user-interface) program.
- You can set the `host` property to be an internet address or hostname. In this case, APLX will attempt to contact an APLX or APLX64 (i.e. 32-bit or 64-bit) Server program on the named system, and ask it to create the new APL task. If the host name is 'localhost', this will be the same physical machine as the Client program. Note that the APLX Server program must already be running on the specified machine.
- You can leave the `host` property as its default value. This will be the same as the `host` property of the parent APL task. This means that the new task will be started on the same system, under the same 32-bit or 64-bit interpreter, as the APL task which created it.

You must set the `host` property before calling the `Open` method, but you can read it back at any time. If you are not running a Client-Server implementation of APLX, the `host` property can only be an empty string.

System Object

For the System object, the `host` property is read-only. It will be an empty string if the current APL task is running locally on the Client machine, or if you are not running a Client-Server implementation of APLX. It will be `'localhost'` if the current APL task is running under an APLX Server on the same machine as the Client. Otherwise, it will be the IP address or host name of the Server on which the current APL task is running.

The 'html' property

Read-only, Character Vector

Valid for: GetMail

The `html` property contains the formatted text of an e-mail, as HTML, if this is available. It is a simple character vector, possibly with embedded carriage return (␣) characters. If the e-mail was sent in accordance with Internet standards, the plain (unformatted) text of the message should be available in the `body` property.

Messages may be sent in plain text only, in which case this property will be an empty vector.

This property is read-only, and is valid only after you have called the `GetMessage` method to retrieve an e-mail from the POP3 server.

The 'icon' property

An integer scalar

Valid for: Icon

The `icon` property determines the icon that displays. It is an integer scalar which corresponds to the ID of the icon. Standard values which are built into the system are:

0	Stop icon
1	Alert icon
2	Note icon
3	Question icon

Under MacOS, you can also create your own icons using a standard tool such as ResEdit, in which case you allocate the resource ID number. When you set this property, APLX looks in the resource fork of the APL program, and then in the system file, to find the icon, looking first for a color icon of type 'cicn', and, if it does not find one, looking for a monochrome icon of type ICON.

The 'id' property

Read-only, Character Vector

Valid for: GetMail

The `id` property contains the unique ID of an e-mail, as a character vector. It is a read-only property, and is valid only after you have called the `GetMessage` (or `GetSummary`) method to retrieve the e-mail from the POP3 server.

The 'imagealloc' property

Implemented under Windows only

Two-element numeric vector

Valid for: ImageList

Sets the initial and increment allocation sizes for the ImageList (i.e. the number of images which will be reserved when the ImageList is created or enlarged). This property is implemented mainly for compatibility with other APL systems.

The 'imagecount' property

Read-only, integer scalar

Valid for: Image, ToolButton, ImageList (*Not implemented under MacOS*)

Contains the number of images in an Image or ImageList, or the number of images in the `bitmap` of a ToolButton.

The 'imageindex' property

Integer scalar

Valid for: Image, and (*except under MacOS*), for Menu, Page

If you have defined an ImageList for a pop-up Menu, window Menu, or Selector control, the `imageindex` property determines (for each child Menu item or Page control) the particular image in the list which will be displayed. Its value is the index (in origin 1) of the image within the list. A value of 0 means no image is displayed. It is not available under MacOS.

For an Image object, the `imageindex` property determines the index of the currently-active image, starting at 1.

The 'imagelist' property

Not implemented under MacOS

Character vector

Valid for: Menu, Selector, Tree

The `imagelist` property allows you to associate an ImageList object with a pop-up Menu, a Selector control, or a Tree control. (To associate an ImageList with the menus in a window's menu bar, use the `menuimagelist` property instead.)

For a pop-up Menu or Selector object, you use the `imageindex` property of the child Menu items or Page controls to cause a particular image from the ImageList to be displayed on an individual menu item or tab sheet.

For a Tree control, the fourth column of the `list` property determines which images from the ImageList are displayed for each node. (A Tree control can also have a second ImageList, displayed to the left of those associated with the `imagelist` property. See `imagelistuser`).

The 'imagelistuser' property

Not implemented under MacOS

Character vector

Valid for: Tree

The `imagelistuser` property allows you to specify the a set of images, to be displayed to the left of the nodes of a Tree. (The `imagelist` property also associates a set of images with a Tree. These images are displayed between the `imagelist` images and the node name). The fourth column of the `list` property determines which images from the two ImageLists are displayed for each node.

The 'imagenames' property

Not implemented under MacOS

See below for data format

Valid for: ImageList

The `imagenames` property assign images to the ImageList object. It accepts any of the following data forms:

- A character vector giving an image file name (a bitmap, cursor, or icon file).
- An integer matrix of N by M pixel `COLORVAL` color values.
- A pair of character vectors, the first giving an image file name, the second giving a transparency mask image file name.
- A pair of data items, the first a character vector giving an image file name, the second indicating which color should be regarded as transparent. The format of the second element can be either `('color' COLORVAL)` or `('color' R G B)`.
- A pair of integer matrices, the first giving pixel color values, the second giving a transparency mask.
- A pair of data items, the first an integer matrix of pixel color values, the second indicating a transparency color in the form `('color' COLORVAL)` or `('color' R G B)`.
- A nested vector of multiple instances of the above.

In the above descriptions, R , G and B are individual Red, Green and Blue color values, each in the range 0 to 255. `COLORVAL` is a single integer encoding the color as $256 \times \text{Blue} + \text{Green} + \text{Red}$.

A single bitmap can optionally contribute multiple images to the `ImageList`, laid side by side. To do this, you need first to set the `imagesize` property to be the size of the image. Any bitmap of a width N times the width specified in `imagesize` will then contribute N images to the list.

Note: The `imagenames` property and the `Addimages` method are alternative ways of setting the list of images. The `imagenames` property is persistent, and is used to refill the image list after any change which would normally clear it (such as setting the `imagesize` property).

The 'imagesize' property

Two-element numeric vector

Valid for: `Image`, `Picture`, `Grid`, `ImageList` (*ImageList not implemented under MacOS*)

For an `ImageList`, the `imagesize` property determines the size of the images in the list (they all have the same size). The first element is the height, and the second the width, in the current `scale` of the `ImageList`.

Note: If you change this property, any images you have added using the `Addimages` method will be removed from the `ImageList`.

For a `Picture` or `Grid`, the `imagesize` property holds the size (in current `scale` units) of the image displayed in the control. This may be larger than or smaller than the `size` property of the object. If it is larger, and the `style` property is set to show scroll bars, the user can use the control's scroll bars to scroll the image within the visible area.

For a `Grid` object, this property is read-only. It is equivalent to the sums of all the row and column sizes in the `Grid`.

For an `Image` object, this property contains the size of the image. If you change it, the image will be rescaled to the new size.

The 'increment' property

Integer scalar (or two-element vector for a Trackbar)

Valid for: Progress, Spinner, Trackbar

The `increment` property contains the amount by which the control's value will change when the user clicks on one of the buttons of a Spinner control, or when the `StepIt` method is called for a Progress bar. For a Trackbar, it is a two-element vector; the first element is the 'page' increment, and the second is the 'line' increment.

The 'indent' property

Numeric scalar

Valid for: Tree

The `indent` property determines the amount by which successive levels in the tree view are indented. It is expressed in the current horizontal `scale` units for the Tree control.

The 'interval' property

Integer scalar

Valid for: Timer

The `interval` property determines the frequency (in milliseconds) at which a Timer object will invoke its `onTimer` callback. The accuracy of the interval is not exact, and the resolution of the timer depends on the host system. The default value is 1000, giving an interval of approximately one second.

The 'labeledithwnd' property

Implemented under Windows only

Read-Only, Integer Scalar

Valid for: Tree

The `labeledithwnd` property is used for low-level programming only. Its value is the internal value which identifies the edit control when the user is editing tree labels.

It can be passed to external calls which require a handle argument via `QNA` or Auxiliary Processors.

The 'limit' property

Integer scalar (numeric scalar for Splitter)

Valid for: Edit, RichEdit, Combo, Splitter

For an Edit, RichEdit or Combo control, the `limit` property determines the maximum number of characters which the user can type into the control. If this value is 0 (which is the default), there is no application-defined limit.

For a Splitter, the `limit` property determines the minimum size of panes, in the current object's scale.

The 'linecount' property

Read-only, Integer scalar

Valid for: RichEdit

The `linecount` property contains the number of lines in a RichEdit control.

The 'lineheight' property

Numeric scalar

Valid for: Printer

When you print multiple lines of text using the `Print` method of a `Printer` object, APLX usually separates the lines by the standard line height associated with the current font. This behavior applies if the `lineheight` property is set to its default value of `-1`

If you want to specify a different line spacing, you can set the `lineheight` property to be a positive value, expressed in the current `scale` of the object.

The 'linetype' property

Integer Scalar

Valid for: Series

Normally, lines associated with a `Series` are drawn as a continuous (solid) line. If you wish, you can specify a different line type for the series, by assigning one of the following values to the `linetype` property:

1=Solid, 3=Dash, 4=Dot, 5=DashDot, 6=DashDotDot

Note: If the `Chart`'s `monochrome` property is set to 1, and you haven't specified your own line type pattern, a default type from one of the above will be assigned to the `Series` object to distinguish it from other series on the chart.

The 'linewidth' property

Integer Scalar

Valid for: Chart, Series

The `linewidth` property of a Chart object specifies the width in pixels of lines drawn in the chart. The default is 1. The value set here is used for all lines (axes, grids, and the data series lines), unless you specify a different value for a particular element using one of the more specific properties `axiswidth` or `gridwidth`, or the `linewidth` property of a specific Series.

For a Series object, this property determines the width (in pixels) of lines drawn for the specific series. By default, it is `-1`, which means that the width should be taken from the `linewidth` property of the parent Chart object. By assigning a different value, you can specify a different line width for this series only.

The 'list' property

A character vector or matrix, or a nested matrix for a Tree

Valid for: List, Combo, Tree

List and Combo

The list property specifies the list of items displayed in a List box or Combo box. You can specify it as either a character matrix or a character vector (with lines delimited by carriage returns). For example, to set the contents of a list box to be the days of the week, you could enter:

```
MyWin.List.list←␣W
```

When referenced, the `list` property returns a character matrix.

Tree

For a Tree object, the `list` property is a matrix of 3, 4 or 5 columns, with one row for each node. (If there is just one node, it can be a vector).

Each row has fields: id depth label [image_indices [state]]:

```

|-----|
|>-----|
| 0 1 | LABEL1 | | 1 1 0 0 | |>-----| |>-----| | | | |
|     | L-----| | L~-----| | | HIGHLIGHTBOLD | | EXPANDED | |
|     | |-----| | |-----| | | L-----| | L-----| |
|     | |-----| | |-----| | | L-----| | |
|     | |-----| | |-----| | | L-----| | |
| 0 2 | LABEL1_A | | 1 1 0 0 | | HIGHLIGHTBOLD | |
|     | L-----| | L~-----| | | L-----| | |
|     | |-----| | |-----| | | L-----| | |
| 0 1 | LABEL2 | | 1 1 0 0 | | EXPANDED | |
|     | L-----| | L~-----| | | L-----| | |
|-----|
|L-----|

```

This list is essentially a flattened hierarchy where the level of each node is given by the (relative but dense) 'depth' field. See the description of the Tree object for more details.

The 'lowvalues' property

Numeric Vector

Valid for: Series

The `lowvalues` property sets the data used for the low values of a Series object displayed on a High-Low-Open-Close or Candlestick chart. It is a numeric vector.

You should normally provide the same number of values for each of the `openvalues`, `closevalues`, `highvalues` and `lowvalues` properties. However, the `openvalues` and `closevalues` properties are optional, and can be left as an empty vector.

The 'margin' property

Four-element numeric scalar

Valid for: Printer

The `margin` property is a four-element vector containing the Left, Right, Top and Bottom margins for printing in the current print job, in the current `scale` units for the Printer object.

The 'marker' property

Character Scalar (or empty vector)

Valid for: Series

When a series is drawn as a Scatter chart, each point is denoted by a marker symbol. By default, the symbol to use is selected when the series is created. By changing the `marker` property, you can choose a particular marker symbol. In addition, if you specify a marker using this property, the marker will be also be displayed if the series is shown as a Line chart.

The marker is specified by assigning a character scalar, which is one of the following APL symbols: + × * o φ e ∇ Δ ◇ □ ↑ ↓ → ← It can also be a space or empty vector, in which case no marker is drawn.

Note: The actual marker is drawn is not a character, but a shape similar to that of the corresponding APL symbol. The APL font is not used to render the marker shape.

The 'maskcolor' property

Not implemented under MacOS

Integer scalar or 3-element vector

Valid for: Tree

When you specify an image in an ImageList object using the `imagenames` property or `Addimages` method, you can provide a mask which indicates that certain portions of the image should be transparent. The `maskcolor` property provides an alternative way of achieving a similar effect. As new images are added, and provided no explicit mask is specified for a given image, any pixel in the bitmap which matches the color specified in `maskcolor` will be set to be transparent.

You can specify `maskcolor` as a three-element vector of Red, Green and Blue color values, each in the range 0 to 255, or as a single integer encoding the color as $256 \times \text{Blue} + \text{Green} + \text{Red}$.

The 'maxsize' property

Two-element numeric vector

Valid for: All displayable controls, but usually used for window-level objects (Form, Dialog, Document, Window). Under MacOS, valid only for window-level objects.

When you create a resizable window, you may want to limit the maximum size to which the user can enlarge it. The `maxsize` property allows you to define the upper limit. The first element is the maximum height, and the second is the maximum width, in the current `scale` of the object. A value of 0 means no constraint.

See also the `minsize` property.

The 'menuimagelist' property

Not implemented under MacOS

Character vector

Valid for: Form, Dialog, Document, Window

The `menuimagelist` property allows you to associate an `ImageList` object with a window's menu bar. You can then use the `imageindex` property of the individual `Menu` items on the menu bar or in sub-menus to cause a particular image from the `ImageList` to be displayed next to the menu item.

To associate an `ImageList` with the menus in a pop-up menu (rather than a window's menu bar), use the `imagelist` property of the menu instead.

The 'messages' property

Read-only, Nested Matrix

Valid for: GetMail

Once you have established a connection to a POP3 mail server using the `Open` method of a `GetMail` object, you can retrieve a list of the messages on the server by reading the `messages` property. This is a nested matrix, with one row per message.

The columns (in Index Origin 1) are:

1. The 'From' field, i.e. the sender of the message, as a character vector.
2. The 'To' field, i.e. the recipient or recipients of the message, as a character vector. If there are more than one, they will be delimited by commas.
3. The 'Date' field, as a character vector, in the format 'Fri, 14 Jan 2005 10:31:00 +0000'.
4. The 'Subject' field, as a character vector.
5. The Size of the message in bytes, as an integer.
6. A Boolean scalar indicating whether the message has multiple parts.
7. The unique Message ID, as a character vector.

The messages are listed in order, so you can retrieve them by index position using the `GetMessage` or `GetSummary` methods. Message indices always start at 1, so the e-mail associated with the first row of the `messages` matrix can be retrieved as message number 1.

The 'methods' property

Read-only, Nested vector of character vectors

Valid for: All objects

The `methods` property returns a nested vector of the names of the methods which are valid for a particular object.

Under Windows, for an OCX/ActiveX control, OLEContainer, or OLE Server Application, methods defined by the external software will be included in the list, preceded by 'X'.

The 'minsize' property

Two-element numeric vector

Valid for: All displayable controls, but usually used for window-level objects (Form, Dialog, Document, Window). Under MacOS, valid only for window-level objects.

When you create a resizable window, you may want to limit the minimum size to which the user can shrink it. Typically, this is to ensure that buttons and other controls are still usable. The `minsize` property allows you to define the lower limit. The first element is the minimum height, and the second is the minimum width, in the current `scale` of the object. A value of 0 means no constraint.

See also the `maxsize` property.

The 'modified' property

Boolean scalar

Valid for: `OLEContainer` (*Implemented under Windows only*)

The `modified` property is set to 1 whenever a document in an `OLEContainer` control is modified by the user. You can set it back to 0, in which case it will remain 0 until the document is modified again.

The main purpose of this property is to indicate to your program that it is necessary save the contents of the document in a file (you can do this by reading the `contents` property and writing it out either to a native file or an APL component file).

The 'monochrome' property

Boolean Scalar

Valid for: Chart

Normally, charts are drawn in color, with different series being drawn in different colors to distinguish them. Sometimes, however, you may want the chart to be drawn in monochrome, for example if you are printing it on a black-and-white printer. The `monochrome` property allows this. If you set it to 1, all the elements of the chart will be drawn in the Chart object's Foreground color (by default, black).

Since this means that the different series cannot be distinguished by color, they will instead be distinguished by line type (solid, dashed, dotted, etc) or by fill pattern (vertical hatch, horizontal hatch etc). See the `linetype` and `fillpattern` properties of the Series object for more information on these.

The 'movieref' property

Read-Only, Integer scalar

Valid for: Movie

The `movieref` property is used for low-level programming only. Under MacOS, it contains the internal Quicktime movie reference ID. Under Windows, it contains the Windows media player handle. It may be zero if the movie file has not been opened successfully.

By using the value contained in the `movieref` property in an Auxiliary Processor or `⌈NA` call, it is possible to exercise closer control over the movie than is supported by the standard Movie object, but this requires knowledge of low-level programming.

The 'name' property

Character Vector

Valid for: All objects

If you created the object using `⌈NEW`, the `name` property is the internal name of the object, allocated by APLX. You can use this name as the right argument of `⌈WE`, to wait for events until a window has been closed:

```
win←'⌈' ⌈NEW 'Window'
win.caption←'Sample Window'
win.name
SYSOBJ-3
⌈WE win.name
```

If you created the object using `⌈WI`, the `name` property is the which you specified as the left argument, excluding the parent name if any. In this case you can change the name subsequently, as in this example:

```
'Win1.OK' ⌈WI 'name'           ⌈ Read the current name
OK
'Win1.OK' ⌈WI 'name' 'OKBut'   ⌈ Change the name
'Win1.OKBut' ⌈WI 'name'       ⌈ Note left argument
OKBut
```

If you want the fully-qualified name including the parent hierarchy, use the `self` property instead.

The 'note' property

Character Vector

Valid for: Chart

The `note` property is one of three labels which you can associate with a Chart (the others are `title` and `subtitle`). It is typically displayed in a small font, and is intended to represent additional information such as a copyright notice or acknowledgement. However, you can use it in any way you wish, or omit it altogether by leaving it as an empty vector.

You can specify the position, color and font of the displayed text by using the `placenote`, `colornote` and `fontnote` properties.

The 'offline' property

Boolean Scalar

Valid for: Browser

If the `offline` property is 1, the Browser object will access only local files and cached copies of pages. The default is 0, meaning that the Browser will connect via the Internet to access web pages where necessary.

The 'oleclasses' property

Read-only, 4-column nested matrix of character vectors

Valid for: System object

The `oleclasses` property of the System object returns a nested matrix which gives details of the OLE Server Applications installed on your system. (The same information is also shown in the Control Browser on the Tools menu). An OLE Server Application is a program (such as Microsoft Word or Microsoft Excel) which you can invoke and control from APLX.

The returned array has one row for each OLE server application installed. However, not all of these will necessarily be usable from APLX, since they may not be designed for use from general OLE client programs such as APLX.

The first column is the plain text name, for example 'Microsoft Excel'. The second column is an alphanumeric string enclosed in curly brackets, which is the unique ID of the OCX control. The third column is the object class name, which is usually a period-delimited string giving the vendor name, object class name and optionally the version number, for example 'Excel.Application'. (The fourth column is reserved for future use).

You can use any of these as the class name to identify the control to APLX, but we recommend that you use the third.

Under MacOS and Linux, this property always returns an empty matrix because OLE is not implemented.

Example

Retrieve the list of OLE server applications installed on the system:

```
OLE←'#' ρWI 'oleclasses'
ρOLE
70 4
```

In this case there are a total of 70 OLE server applications installed.

If Microsoft Word is installed on the system, it should appear in the list returned by the `oleclasses` property:

```
OLE[;1]⊖c'Microsoft Word'
2
OLE[2;]
Microsoft Word {000209FF-0000-0000-C000-000000000046} Word.Application
```

You can use any of the three strings to identify the control. The following sequences are all equivalent - they each invoke Microsoft Word and create an APLX object called 'WORD' which has properties and methods associated with it:

(a) Use the plain name to identify the OLE Server Application:

```
WORD←'□' □NEW 'Microsoft Word'
```

(b) Use the unique class ID:

```
WORD←'□' □NEW '{000209FF-0000-0000-C000-000000000046}'
```

(c) Use the object class name (recommended):

```
WORD←'□' □NEW 'Word.Application'
```

The 'oledotypes' property

Read-only, 4-column nested matrix of character vectors

Valid for: System object

The `oledotypes` property of the System object returns a nested matrix which gives details of the OLE document types available on your system. These are classes of document which can be placed in an `OLEContainer` object and thus activated in one of your windows. (The same information is also shown in the Control Browser on the Tools menu).

The returned array has one row for each OLE document type available. However, not all of these will necessarily be usable from APLX, since they may be private document types which are not designed for use by other applications.

The first column is the plain text name, for example 'Microsoft Document'. The second column is an alphanumeric string enclosed in curly brackets, which is the unique ID of the OCX control. The third column is the programmatic class name, which is usually a period-delimited string giving the vendor name, object class name and optionally the version number, for example 'Word.Document'. (The fourth column is reserved for future use).

The third of these is the name which corresponds to the `progid` property of the `OLEContainer` object, which can be used to create an empty document of a particular type.

Under MacOS and Linux, the `oledotypes` property always returns an empty matrix because OLE is not implemented.

Example

Retrieve the list of OLE document types installed on the system:

```
OLE←'#' ⍵WI 'oledoctype'
ρOLE
25 4
```

In this case there are a total of 25 types of OLE document known to the system.

On most Windows systems, one of the valid OLE document types is a Bitmap Image, editable using Microsoft Paint. If it is available on the system, it should appear in the list of classes returned by the `oledoctype` property:

```
OLE[;1]⍵'Bitmap Image'
24
OLE[24;]
Bitmap Image {D3E34B21-9D75-101A-8C3D-00AA001A1652} Paint.Picture
```

We can use this information to create a new bitmap image in an OLE container inside an APLX window:

```
MYWIN←'⍵' ⍵NEW 'Window'
MYWIN.OLE.New 'OLEContainer' ⍵ MYWIN.OLE.align←1
MYWIN.OLE.progid←'Paint.Picture'
```

The 'opened' property

Boolean scalar

Valid for: All objects

The `opened` property is 1 if the object is currently open, and 0 if it is closed. For all objects except the System object, you can set the value of `opened` in order to force the object to open or close. Setting it to 1 is equivalent to calling the `Open` method, and setting it to 0 is equivalent to calling the `Close` method with an argument of 1 (unconditional close).

The 'openvalues' property

Numeric Vector

Valid for: Series

The `openvalues` property sets the data used for the opening values of a Series object displayed on a High-Low-Open-Close or Candlestick chart. It is a numeric vector.

You should normally provide the same number of values for each of the `openvalues`, `closevalues`, `highvalues` and `lowvalues` properties. However, the `openvalues` and `closevalues` properties are optional, and can be left as an empty vector.

The 'order' property

Integer scalar

Valid for: Menu, Check, Radio, Button, Progress, Trackbar, Spinner, Scrollbar, List, Combo, Edit, RichEdit, Movie, Splitter, Selector, Page, Tree, Frame

The `order` property defines the logical position of a child control amongst the controls of its parent. As each control is created, it is assigned the next integer value, starting at 1. You can change the order by assigning a new integer or fractional value; the other controls will be re-ordered accordingly. This can be used to specify the sequence in which controls on a window are selected using the Tab key.

Menu and Page controls are displayed in the sequence defined by the `order` property. For example, in a sub-menu, the first item has `order 1`, the second has `order 2`, etc.

Note: Under MacOS, the `order` property is ignored for Menu controls. Menu items cannot be re-ordered once they have been created.

The 'orientation' property

Boolean scalar

Valid for: Printer

The `orientation` property is 0 for printing Portrait and 1 for Landscape. It is usually chosen by the user in the print Setup dialog. Alternatively, you can set it under program control (you must do this *before* calling the `Open` method).

The 'overlays' property

Implemented under Windows only

Integer vector of up to four elements

Valid for: ImageList

The `overlays` property allows you to specify the index positions (in origin 1) of up to four images in the ImageList which will be used to overlay other images. These overlays can then be used in a Tree object.

The 'page' property

Read-only, integer scalar

Valid for: Printer

The `page` property contains the current page number during a print job. It is incremented automatically whenever a new page is started.

The 'password' property

Character Vector

Valid for: GetMail, SendMail, HTTPClient

Before you can use the GetMail or SendMail objects to retrieve or send mail, you need to call the `Open` method to connect to the mail server which you specify using the `host` property. Depending on how the server is set up, you will probably need to set a user name and password before calling `Open`. The password to use will normally be supplied by your Internet Service Provider (ISP) or network administrator.

For an example, see the description of the `host` property.

For the HTTPClient object, the `password` property should be set to access pages which require a user name and password.

The 'path' property

Character Vector

Valid for: GetMail

When you use the `GetMessage` method of the GetMail object to retrieve an e-mail from the POP3 server, any attachments are saved as files. The `path` property allows you to specify the directory in which you want attachments to be saved. An empty vector means the user's home directory.

The 'pen' property

Synonym for `pensize`

The 'pensize' property

Alternative name: `pen`

Two-element integer vector

Valid for: Line, Rectangle, RoundRect, Arc

The line width in pixels is determined by the `pensize` property. It comprises two integers, representing the height and width of the imaginary 'pen' used to draw lines. The default is 1 1. For example, if you change the pen size for a rectangle like this:

```
Win1.Rect.pensize←1 2
```

then the sides of the rectangle will be twice as thick as the top and bottom lines.

Note: Under Windows and Linux, the height and width of the pen cannot be set independently. You can specify either a single number or two numbers, but only the first is used. If you read the property, a two-element vector is returned but the two elements will be the same.

The 'picture' property

Not implemented under Linux

An integer vector containing graphic data

Valid for: Picture, System, Chart

The `picture` property contains the commands which display the graphic in a Picture object. The commands are encoded in an integer vector, and comprise a PICT or JPEG image (under MacOS) or a Windows Enhanced Metafile (under Windows). You can obtain a picture vector in a number of ways. For example, you can fetch a picture from the Clipboard by reading the `picture` property from the System object. Under MacOS, you can also create your own PICT variable using the functions in the QUICKDRAW workspace. Under Windows, you can read a metafile (`.emf`) from disk, using `DNREAD` to convert it to an integer vector (conversion type 2).

Generally, pictures look best if they are displayed at this original size (or an integral multiple of the original size). You can, however, change the size of the picture to stretch or compress it, but to do this you must change the `size` after setting the `picture` property.

For the System object, the `picture` property represents a graphic image on the Clipboard. Reading the `picture` property returns an integer vector representing the picture currently in the Clipboard, or an empty vector if there is no picture in the Clipboard. Writing an integer vector to the System `picture` property places a picture on the Clipboard. The data written must be a valid graphic, i.e. a PICT or JPEG image under MacOS, or a metafile under Windows.

This Windows example creates a picture object, and displays a metafile image from disk. It then reads the `picture` property and places it on the Clipboard:

```

MYWIN←'□' □NEW 'Window'
MYWIN.PIC.New 'Picture'
FILE←'C:\Program Files\Microsoft Works\workscor\j0160036.wmf'
MYWIN.PIC.file←FILE
METAFILE←MYWIN.PIC.picture
ρMETAFILE
14771 '#' □WI 'picture' METAFILE

```

For a Chart object, the `picture` property is read-only. It contains the drawing commands (vector graphics) which will reproduce the chart on the current platform.

See also the `bitmap` property, which allows you to retrieve or set an image as an array of pixel color values.

The 'pitch' property

Numeric scalar

Valid for: Printer

When you print text using the `Print` method of a Printer object, APLX usually prints each character in its standard width, as set by the current font. This behavior applies if the `pitch` property is set to its default value of `⍒1`

If you want to specify a different character spacing, you can set the `pitch` property to be a positive value, expressed in the current `scale` of the object. This will then be used as the fixed width of each character cell.

The 'placelegend' property

Character Vector

Valid for: Chart

The `placelegend` property determines where the chart legend is drawn. The legend shows, for each series which has a non-empty `caption` (or `title`) property, a sample line, marker, or filled rectangle as appropriate, next to the caption. (If a given series has no caption, it will not appear in the legend). This property is a text string, which can be one of the values 'top', 'left', 'right', 'none', or an empty vector. If it is 'none' or an empty vector, the legend is not shown. The default is 'top'.

The 'placenote' property

Character Vector

Valid for: Chart

The `placenote` property determines where the Chart's note (if any) is drawn. It is a text string, and can be one of the values 'top', 'topleft', 'topright', 'bottom', 'bottomleft', 'bottomright'.

The default is 'bottomright'.

The 'placetitle' property

Character Vector

Valid for: Chart

The `placetitle` property determines where the Chart's title (or caption) and subtitle (if any) are drawn. It is a text string, and can be one of the values 'top', 'topleft', 'topright', 'bottom', 'bottomleft', 'bottomright'.

The default is 'top', meaning the title is centered above the chart.

The 'playing' property

Read-only: Integer scalar

Valid for: Movie

The `playing` property is a read-only property which allows you to find out whether a movie is playing (see also the `onMovieEnd` callback which can be invoked when a movie ends). The possible values returned are:

-2	Movie file selected, and playing is handled by a controller.
-1	Movie file not yet specified (or user chose Cancel in the movie selection dialog, or an error occurred in trying to open the file)
0	Movie file selected but not yet playing (either not started or rewound under program control)
1	Movie playing or stopped in the middle of the movie
2	Movie ended

The 'pointer' property

Valid for: All visible controls, and the System object

The 'pointer' property determines the type of mouse cursor which will display over the control. It is an integer scalar, one of:

0:	Default
1:	Arrow
2:	Cross
3:	IBeam
5:	Size
6:	SizeNESW
7:	SizeNS
8:	SizeNWSE
9:	SizeWE
10:	UpArrow
11:	HourGlass
12:	No
13:	AppStart
14:	Help
30:	Drag
31:	NoDrop
32:	HSplit
33:	VSplit
34:	SQLWait
35:	HandPoint
100-119:	User-defined pointer (See the <code>Loadpointer</code> method of the System object for more details).

If you change the `pointer` property of the `System` object, the cursor will change wherever it is on the screen.

The 'port' property

Integer Scalar

Valid for: `Socket`, `GetMail`, `SendMail`, `APL (Child task)`

The `port` property determines the TCP/IP port used for communication by a `Socket` object. You need to set it before calling the `Open` or `Listen` methods.

Port numbers are usually associated with specific protocols. For example, port 80 is used for the HTTP protocol used by web browsers.

The `port` property can also be used to alter the default port used by the `GetMail` and `SendMail` objects. You need to set it before calling the `Open` method.

For an `APL (Child Task)` object in a Client-Server version of APLX, the `port` property determines the TCP/IP port number used to communicate between the Client and the APLX Server. Normally you should leave this at the default value 1134, the port number allocated to APLX by IANA, the Internet Assigned Numbers Authority. If, exceptionally, your APLX Server is listening on a different port number, you can set this property before opening the task.

The 'position' property

Two-element numeric vector (Integer scalar for `Movie` object)

Valid for: `Printer`, `Movie`

For a `Printer` object, the `position` property is a two-element numeric vector containing the Y and X print position, in the current `scale` units. The position is relative to the top, left margins of the page as set by the `margin` property. It is updated automatically as you print text using the `Print` method, or if a page throw occurs. You can change the `position` property to cause printing to occur at a particular place on the page.

For a `Movie` object, the `position` property is an integer scalar containing the point reached in the movie. This will be a value between the two elements in the `range` property.

The 'printername' property

Currently implemented under Windows only

Character vector

Valid for: Printer

The `printername` property contains the device name of the currently-selected printer. Usually this can be chosen by the user in the printer dialogs (see the `Setup` and `Job` methods). Alternatively, you can select the printer under program control by writing the name to the `printername` property (you must do this *before* calling the `Open` method). If you write an empty vector, the system default printer is selected.

The `printers` property contains the list of available printers.

The 'printers' property

Not implemented under MacOS

Read-only, Nested vector of character vectors

Valid for: System and Printer

The `printers` property allows you to determine the names of the printers available on the system on which APLX is running. It returns a nested vector, where each element is a character vector containing the name of a printer.

Under Windows, you can use the `printername` property to set or query which of these printers is currently selected for printing.

The 'progid' property

Character vector

Valid for: OLEContainer, OCX controls, and OLE server application (*Implemented under Windows only*)

The `progid` property contains the programmatic class name of the external OCX control or application.

For an OLEContainer, `progid` refers to the class of the document in the container (for example, whether it is a Word document or a Pinnacle Chart). It is of the same form as the third column of the System object's `oledoctype`s property. If there is a document loaded, reading the `progid` property allows you to find out what the document type is. Writing to the `progid` property creates a new empty document of the appropriate class. If you write an empty vector, a dialog is displayed allowing the user to choose.

For an OCX control or OLE server application, `progid` is a read-only property. It returns the programmatic class name in the same form as the third column of the System object's `xclasses` or `oleclasses` property.

The 'properties' property

Read-only, Nested vector of character vectors

Valid for: All objects

The `properties` property returns a nested vector of the names of the properties which are valid for a particular object.

Under Windows, for an OCX/ActiveX control, OLEContainer, or OLE Server Application, properties defined by the external software will be included in the list, preceded by 'x'.

The 'protocol' property

Integer scalar

Valid for: Socket

The `protocol` property determines the protocol used by the socket. For details, see the documentation on Windows or Unix sockets.

The default is 0, meaning the protocol used should be the default for the address family selected using the `family` property. You normally do not need to change this.

The 'proxy' property

Nested Vector

Valid for: HTTPClient

If you are accessing the Internet from a corporate network, you may have to use a 'proxy' server when you retrieve web pages. This property allows you to specify the proxy server, as a nested vector of (Proxy Address) (Port). The Proxy Address is a character vector (the Internet address of the server), and the Port is an integer scalar (the TCP/IP port to use). Your Network Administrator should be able to provide you with the values you need for these parameters.

If you set this property to an empty vector, no proxy server will be used. This is the default.

The 'quality' property

Integer Scalar

Valid for: Image

Some image formats allow you to specify an image quality parameter, where higher values give a better-quality image but require a larger file size. This property allows you to set or retrieve the quality parameter. The exact meaning depends on the image `format`. See the ImageMagick documentation for more details.

The 'range' property

Two-element numeric vector, or empty vector

Valid for: Progress, Spinner, Trackbar, ChooseFont, Movie, RichEdit

For a Progress, Spinner or Trackbar control, the `range` property is a two-element integer vector. The first element is the minimum value of the control, and the second is the maximum value. Writing an empty vector this property restores the default value: 0 100.

For a ChooseFont pre-defined dialog under Windows, the `range` property allows you to set the smallest and largest font sizes which the user can select (in the current `scale` units). An empty vector indicates that there is no pre-set limit on the font size. Remember that the default scale is character units - you will normally find it easier to select `scale 3`, point units, for the ChooseFont dialog.

For a Movie object, `range` is a read-only property. It returns the start position (usually 0) and end position of the movie or other media file, in units which depend on the particular type of file. The `position` property represents the current position within this range.

For a RichEdit control, the `range` property is read-only, and is a two-element integer vector. The first element is the line number of the first visible line, and the second is the count of lines visible in the control.

The 'referrer' property

Character Vector

Valid for: HTTPClient

This property contains the Referrer URL which will be passed to the web server when you access a page using the HTTPClient object. It is typically used by web administrators to keep track of the page which contained the link by which a given user reached the web-site.

The 'replyto' property

Character Vector

Valid for: GetMail, SendMail

The `replyto` property contains the e-mail address to which any reply to an e-mail should be sent. It is a simple character vector.

For the SendMail object, you should set this property before calling the `SendMessage` method. The default is an empty vector, which means that the reply address is the same as the 'From' address.

For the GetMail object, this property is read-only, and is valid only after you have called the `GetMessage` method to retrieve an e-mail from the POP3 server.

The 'rounding' property

Length 2 integer vector

Valid for: RoundRect

The rounding property determines the curvature of the corners of a round rectangle, in the current scale of the object. The first element is the vertical rounding, and the second is the horizontal rounding. The default is 8 pixels for both elements.

The 'row' property

Integer Scalar

Valid for: Grid

The Row number (in index origin 1) of the currently-active cell. You can write to this property to change the current cell.

The 'rows' property

Integer Scalar

Valid for: Grid

The number of data (scrolling) rows in the Grid control. You typically set this property when you create the Grid to fit the size of the data you want to display.

The 'rowsize' property

Valid for: Grid (*Special syntax applies*)

You can set the size of the rows of a Grid control using the syntax:

```
WindowRef.ControlName.rowsize Rows Sizes
```

or:

```
'WindowName.ControlName' □WI 'rowsize' Rows Sizes
```

where Rows is a scalar or vector list of row numbers, and Sizes is a matching vector (or scalar) of the row sizes, in pixels. A row number of 0 sets the default size for all rows which are not explicitly set.

Note: At least one of the Rows and Sizes parameters must be a vector, otherwise the statement will be interpreted as an attempt to read back the current sizes.

You can read back the current sizes using the syntax:

```
R ← WindowRef.ControlName.rowsize Rows
```

or:

```
R ← 'WindowName.ControlName' □WI 'rowsize' Rows
```

Note that, depending on the flags you have set in the `style` property, the user may be able to resize the rows.

The 'rtf' property

Implemented under Windows only

Character vector

Valid for: RichEdit

Internally, Windows RichEdit controls hold information about the contents and appearance of text in Rich Text Format (RTF). The `rtf` property allows you to retrieve or change the entire contents of the control as RTF directly, including the text contents, font face, size, style and color.

If you want just the raw text without any formatting information, use the `text` property instead. If you want to retrieve or change just the currently-selected text in RTF format, use the `selrtf` property.

The 'scale' property

Scalar integer in the range 1 to 6

Valid for: All displayable objects, and the Printer and System objects

The `scale` property sets the units which apply to dimensions of an object. The supported values are:

1. Use 'character' units. A character unit has the same height as a line of text in the default system font (16 points), and the width of a typical character (8 points). However, because the system font is a proportional font, the width of a given string of characters is not in general an exact multiple of character units. Note that this case has the unfortunate characteristic that the vertical and horizontal scales are different, so it is best to avoid using it.
2. Use 'dialog' units, (1/8 system-font high, 1/4 system-font wide)
3. Use point units (nominally 72 points per inch)
4. Use 'twips' units (nominally 1440 twips per inch)
5. Use pixel units (also usually 72 per inch on the Macintosh), but this depends on the resolution and screen size.
6. Use nominal millimetres. Depending on the screen size, this may not be accurate for controls displayed on a screen, but it should be accurate when printing on paper (provided the printer driver is set up correctly).

Changing the `scale` property on an object has no immediate visible effect, but sets the units for subsequent reading or setting of dimensional properties such as `size`. It is important to remember that the `size` property uses the units of the object to which it applies, but the `where` and `extent` properties use the units of the *parent* object. For compatibility with other APL systems, the default value for `scale` is 1, character units. However, it is usually much better to use `scale 5`, pixel units, in your applications.

Note that, in APLX, the parent control's `scale` property is inherited by any children at the time the child is created. Thus, if you set the `scale` of the System object to 5 before creating any windows, all subsequently-created objects will inherit this value.

Scale used for the Draw method

When you use the `Draw` method to draw geometric shapes and text on a control or window, a separate scale (specific to the `Draw` method operations) is used. This is initially set to be the same as the control's `scale` property value, but can be changed using the 'Scale' operation of the `Draw` method. This does not affect the `scale` property of the control.

In addition to the values shown above, you also have the option of using a Draw-method scale which is proportional to the window or control size.

The 'searchstring' property

Implemented under Windows only

Read-only, character vector

Valid for: Tree

As the user uses the keyboard to enter the first few characters of a name in a Tree object, the `searchstring` property contains the string which has been typed and which will be used to find the node.

The 'selalign' property

Character vector, one of 'left' 'right' 'center' (or 'centre')

Valid for: RichEdit

The `selalign` property determines whether the paragraph containing the current selection point is left-aligned, right-aligned, or centered within the control.

The 'selbullet' property

Character vector, one of 'none' or 'bullet', or a boolean scalar

Valid for: RichEdit

The `selbullet` property determines whether the paragraph containing the current selection point is has a 'bullet point' indicator. You can set this property to 'bullet' or 1 to switch on the bullet, or 'none' or 0 to switch it off.

The 'selcolor' property

Color value, expressed as an integer scalar or 3-element vector

Valid for: RichEdit

The `selcolor` property determines the foreground color of the selected text. Colors can be specified in one of two ways. In the first way, you specify a vector of three integers (Red, Green and Blue), each in the range 0 to 255. For example 255 0 0 is pure red and 255 255 255 is white. In the second way, you specify a single integer which encodes the three values in base 256 as $256 \cdot \text{Blue} + \text{Green} + \text{Red}$. In addition, two special values are recognised; -1 means use the parent's foreground or background color as appropriate, and -2 means use the default color for the rich-edit control's text (usually black).

When you read this property, APLX returns the single-integer form.

The 'selection' property

A two-element integer vector, or four-element integer vector for a Grid

Valid for: Edit, RichEdit, Document, Combo (*except under MacOS*), Grid

For an Edit, RichEdit, Combo or Document object, the `selection` property determines the position of the insertion point and the length of the selected (highlighted) text. The first element is the position of the insertion point (or start of the selected text) in index origin 0. The second is the length of the selection, or 0 if there is none. Setting the selection length to `-1` selects from the start position to the end of the text.

You can set this property under program control, or it can be changed by the user (you can detect that this has happened by means of the `onClick` callback).

For a Grid object, the `selection` property is a four-element integer vector, comprising the Row and Column of the top-left of the selected cell range, followed by the number of rows selected and the number of columns selected. If the `style` property is set so that only single-selection is possible (which is the default), the last two elements will both be one. You can write to this property to change the selection under program control.

The 'self' property

Read only, character vector

Valid for: All objects

The `self` property returns the fully-qualified name of an object, including the parent (or parents) separated by period. This contrasts with the `name` property which returns just the object name without the parent(s).

Note: The `self` property is unusual in that it can be referenced for a non-existent object without giving an error. In this case, it returns an empty vector. It can thus be used to test for the existence of a particular object.

The 'selfont' property

Nested vector or character vector

Valid for: RichEdit

The `selfont` property represents the font of the current selection in a RichEdit control. This allows you to set the font of a particular existing part of the text, or to set the font for text which will subsequently be inserted.

See the description of the `font` property for details on how the font is described.

The 'selindents' property

Three-element numeric vector

Valid for: RichEdit

The `selindents` property contains the left, 'bullet' and right indents for the currently-selected paragraph, in the current `scale` units for the control.

The 'selrtf' property

Implemented under Windows only

Character vector

Valid for: RichEdit

Internally, Windows RichEdit controls hold information about the contents and appearance of text in Rich Text Format (RTF). The `selrtf` property allows you to retrieve or change the currently-selected text as RTF directly, including the text contents, font face, size, style and color.

If you want just the raw text without any formatting information, use the `seltext` property instead. If you want to retrieve or change the entire contents of the control in RTF format, use the `rtf` property.

The 'selstyle' property

5-element boolean (or integer) vector

Valid for: RichEdit

The `selstyle` property contains the style of the currently-selected text as a 5-element vector: bold, italic, underline, strikethrough, and protected. For each element, you can write: 1 = Set, 0 = Clear, -1 = Leave unchanged

Note that the 'protected' attribute means that the text cannot be changed, even under program control. Thus, if you want to use this attribute, you must write out the text first, then select the part you want to protect, and then set this attribute. The 'protected' attribute is ignored under MacOS and Linux. If you want to make the whole of the text read-only, use the `style` property.

The 'seltabs' property

Implemented under Windows only

Numeric vector of up to 32 elements

Valid for: RichEdit

The `seltabs` property determines the tab-stop positions for the currently-selected paragraph, expressed in the `scale` units of the control.

Under MacOS and Linux, the tab stops of a rich-edit control are fixed at four-character intervals.

The 'seltext' property

Character vector

Valid for: Browser, Edit, RichEdit, Document, Combo (*except under MacOS*)

When you read the `seltext` property, the contents of the currently-selected text will be returned as a character vector. If there is no current selection, an empty vector will be returned.

When you write to the `seltext` property, the current selection (if any) will be erased, and the text you write will be displayed in the edit control at the selection point. For a RichEdit control, this text will be formatted according to the `selfont`, `selcolor`, `selalign`, `selbullets` and `selindents` properties.

The `seltext` property is read-only for a Browser object.

The 'separator' property

Boolean scalar

Valid for: Menu

The `separator` property is 1 if this menu item is a separator (i.e. a line which is not selectable), and 0 otherwise.

The 'serverreply' property

Read-Only, Character Vector

Valid for: GetMail, SendMail

When you have called the `GetMessage`, `GetSummary` or `SendMessage` method, the `serverreply` property will contain the reply sent by the server. If all is well this will usually begin "+OK...". It can comprise several lines, delimited by carriage-return characters. It is mainly useful for diagnosing problems with the mail protocol.

The 'shortcut' property

Two-element vector

Valid for: Menu

The `shortcut` property is used to make a menu item selectable via a shortcut key. The first element is the virtual key code of the key, or the character corresponding to the key. For the main letter and number keys, the virtual key code is the same as the ASCII code for the key, eg 65 for 'A' and 49 for '1'. You can specify either the virtual key code as an integer, or (for keys on the main keyboard) as a character scalar.

Under Windows and Linux, the second element determines the modifier keys which apply: 0 (Plain), 1 (Shift), 2 (Ctrl), 4 (Alt) and sums thereof. If you omit the second element, the default is 2 (Ctrl). *Note: Some of these modifiers may not be useful in practice because the operating-system intervenes before APLX sees the keystroke; in Windows, for example, this happens with the Alt key.*

Under MacOS, the second element is ignored since the Command key is always used for menu shortcuts.

Thus, these three lines all do the same thing, and specify Ctrl-C as the shortcut (or Cmd-C under MacOS):

```
DEMO.EditMenu.Copy.shortcut←'C' 2
DEMO.EditMenu.Copy.shortcut←67 2
DEMO.EditMenu.Copy.shortcut←'C'
```

APLX automatically changes the displayed text in the menu to show the shortcut key.

When you read the `shortcut` property, APLX always returns the two-integer form.

The 'size' property

Two-element numeric vector

Valid for: All displayable objects and System

The `size` property determines the height and width of an object. The units depend on the setting of the object's `scale` property, but default to Character units (16 pixels high and 8 pixels wide), which are the height and typical width of a character in the System font. The first element of the `size` property is

the height, and the second the width of the object. (See also the `extent` property, which is the same except that it is expressed in the scale of the object's *parent*.)

This example resizes an Edit box to be 1.5 units high and 20 wide:

```
MyWin.Ed.size←1.5 20
```

The precise meaning of the `size` property depends on the object class, but in general is the size of the enclosing rectangle. For a `Line`, the `size` property determines the offset from the start point to the end point of the line.

For the `System` object, the `size` property is read-only, and returns the height and width of the screen. See also the `workarea` property which returns the rectangle within the screen size which is available for application use.

The 'sizemode' property

Integer scalar

Valid for: `OLEContainer` (*Implemented under Windows only*)

The `sizemode` property determines how the document fits in an `OLEContainer` control. The possible values are:

- 0 = Displays the OLE object at its normal size, clipping any parts that don't fit within the container.
- 1 = Displays the OLE object at its normal size, centering it within the container.
- 2 = Scales or shrinks the view of the OLE object to fit within the container, by scaling width and height proportionally.
- 3 = Scales or shrinks the view of the OLE object to fill the OLE container, without regard to preserving the proportions of the OLE object.
- 4 = Displays the OLE object at its normal size and automatically resizes the container to fit the size of the OLE object.

The default is 0.

The 'sliderlen' property

Implemented under Windows only

Numeric scalar

Valid for: Trackbar

The `sliderlen` property determines the length of the slider part of a Trackbar control, in the current `scale` units. Setting this property to 0 makes the slider invisible.

The 'sliderwhere' property

Implemented under Windows only

Read-Only, Four-element numeric vector

Valid for: Trackbar

This property returns a four element vector giving the top, left, height and width of the slider part of the trackbar, in the current `scale` of the object. The top and left values are relative to the top left corner of the Trackbar control.

The 'sourceformats' property

Valid for: All visible controls

Character vector or nested vector of character vectors

The `sourceformats` property is used to control drag-and-drop operations. It determines the names of the drag-and-drop formats which the control can provide. If there is only one format, it is a simple character vector. If the control can provide more than one format, it is a nested vector of character vectors.

When the user attempts a drag-and-drop operation, the names in the source control's `sourceformats` property are matched against the names in the target control's `targetformats` property. If any of the

names match, the cursor changes to indicate that drag-and-drop is potentially valid, and one or more of the drag-and-drop callbacks (`onDragOver`, `onDragEnter`, `onDragLeave`, `onDragDrop`) are invoked.

Drag-and-drop format names can be anything you choose. However, by convention, you should use `'TEXT'` for controls which support dragging of text.

The 'state' property

Read-only, Nested 2 by N matrix

Valid for: All objects

The `state` property returns a two-column nested matrix. The first column is the name of each property which has been set to a non-default value. The second column is the corresponding value of the property. Read-only properties are not included.

Although you cannot set the `state` property directly, you can use it to re-create or clone an object by passing the property names and values to the `New` or `Set` method of another control.

The 'status' property

Read-only. Integer vector or scalar

Valid for: `Browser`, `GetMail`, `SendMail`, `HTTPClient`, `Socket` and APL (child task) object

GetMail, SendMail, HTTPClient, Socket

For these networking classes, the `status` property is a two-element integer vector. The first element is 0 if the object is not connected, 1 if the object is connected, 2 if it is listening for connections. The second element is the latest error code reported by the underlying operating system.

Browser

For a `Browser` object, the `status` property returns a three-element Boolean vector of (`IsBusy`) (`CanGoback`) (`CanGoForward`).

APL Object

The `status` read-only property of an APL (child task) object indicates the execution state of the task. It returns one of the following codes:

- ^-1 The task is not running (i.e. it has not been opened, or has terminated)
- 0 The task is busy executing an APL expression or command
- 1 The task is in desk-calculator mode, awaiting input
- 2 The task is awaiting \square input
- 3 The task is awaiting \square input
- 4 The task is awaiting \square CC input

The 'style' property

An integer scalar

Valid for: Bevel, Button, ToolButton, Combo, Edit, Frame, Grid, HTTPClient, Image, ImageList, Label, List, Menu, Movie, OLEContainer, Picture, Progress, RichEdit, Scroll, Selector, Spinner, Splitter, Trackbar, Tree, ChooseColor, ChooseFont, OpenFile, SaveFile, MsgBox

The meaning of the `style` property depends on the object. Usually it comprises one or more basic styles, plus a set of optional flags which govern the behavior or appearance of the control.

See the entry for the particular object class for details of how `style` is interpreted for that object.

The 'subject' property

Character Vector

Valid for: GetMail, SendMail

The `subject` property contains the subject of an e-mail, as a character vector.

For the SendMail object, you should set this property before calling the `SendMessage` method.

For the GetMail object, this property is read-only, and is valid only after you have called the `GetMessage` or `GetSummary` method to retrieve an e-mail (or summary of an e-mail) from the POP3 server.

The 'subtitle' property

Character Vector

Valid for: Chart

The `subtitle` property is one of three labels which you can associate with a Chart (the others are `title` and `note`). It is displayed just below the `title`, in a slightly smaller font, and is intended as a sub-title for the whole chart. However, you can use it in any way you wish, or omit it altogether by leaving it as an empty vector.

You can specify the position, color and font of the displayed text by using the `placetitle`, `colortitle` and `fonttitle` properties.

The 'svg' property

Character Vector (read-only)

Valid for: Chart

This read-only property returns the representation of the chart in Scalable Vector Graphics (SVG) format. This format is ideal for publishing your charts, since SVG will retain high-quality output on high-resolution devices.

The 'tabgroup' property

Not currently implemented

The 'tabparent' property

Not currently implemented

The 'tabrows' property

Implemented under Windows only

Read-only, Integer Scalar

Valid for: Selector

Returns the number of rows used for the tabs of a multi-row Selector control.

The 'tabstop' property

Boolean scalar

Valid for: Check, Radio, Button, Progress, Trackbar, Spinner, Scroll, List, Combo, Edit, RichEdit, Movie, Splitter, Selector, Page, Tree, Frame

If the `tabstop` property for a control is 1, the user can move the focus to the control using the Tab key (the sequence is determined by the `order` property, and defaults to the order of creation). If the `tabstop` property is 0, the control cannot be tabbed to.

Note: Under MacOS, only a control which can accept keyboard focus (such as an Edit or RichEdit control) can be tabbed to. This is because of the user-interface conventions which apply in MacOS.

The 'targetformats' property

Valid for: All visible controls

Character vector or nested vector of character vectors

The `targetformats` property is used to control drag-and-drop operations. It determines the names of the drag-and-drop formats which the control can accept if the user drags another control on to it. If there is only one format, it is a simple character vector. If the control can accept more than one format, it is a nested vector of character vectors.

When the user attempts a drag-and-drop operation, the names in the source control's `sourceformats` property are matched against the names in the target control's `targetformats` property. If any of the names match, the cursor changes to indicate that drag-and-drop is potentially valid, and one or more of the drag-and-drop callbacks (`onDragOver`, `onDragEnter`, `onDragLeave`, `onDragDrop`) are invoked.

Drag-and-drop format names can be anything you choose. However, by convention, you should use 'TEXT' for controls which support dragging of text.

The 'taskid' property

Read-only, Integer scalar

Valid for: System, APL (child task) object

Every task has a unique identifier. You can read the task identifier of a child task using the read-only `taskid` property. It is a scalar integer, and can be used in event handling to identify the task:

```
ChildTask.taskid  
53429272
```

The `taskid` property will be zero if the task is not running.

A task can read its own task identifier using the `taskid` property of its System object.

The 'text' property

A character vector

For a Grid, a character vector for each cell

Valid for: Browser, Combo, Edit, RichEdit, Document, System, MsgBox, APL (child task) object
Cell property for Grid object

The `text` property contains the text displayed in an Edit box, Combo, RichEdit control, or Document window, as a character vector possibly containing embedded carriage returns. By setting this property, you can change the contents of the object under program control, but it also changes as a result of user keyboard input (you can detect that this has happened by means of the `onChange` callback).

For a MsgBox, the `text` property determines the text displayed in the dialog.

For an APL (child task) object the text property contains the text in the task's session window, if any. It is a text vector, usually with embedded carriage return (␣R) characters. The `text` property returns the logical contents of the session window, even if the session window is not visible (however it is always an empty vector for a background task, which does not have a session window). You can also write to this property to change the text in the session window of a child task.

For the System object, the `text` property contains the text (if any) in the Clipboard. If there is no text in the Clipboard, it is an empty vector. Writing to this property places (unformatted) text in the Clipboard, ready for pasting into an APLX window or another application.

For a Browser object, the text property is read-only. It contains the text displayed in the Browser, with HTML tags stripped out. (See also the `contents` property, which contains the text including the HTML tags).

Grid object `text` cell property

When you read this cell property for a Grid object, it returns a character vector containing the text displayed in the cell, irrespective of the `cellVs` type, for each cell in the list of rows and columns specified. For multiple cells, it will therefore return a nested array of character vectors.

When you write to this cell property for a Grid object, the text displayed in the cell is set to the text you supply. To set multiple cells in one operation, provide a nested array of character vectors. The type of the cell, and the `cellVs` `valid` property, are not changed. If the cell is Numeric, it will remain so, and its `value` will not change.

Although `text` is a cell property for a Grid object, as a convenience you can also read (but not write to) `text` as a simple property of the Grid object. It returns the text of the currently-active cell.

The 'textalign' property

Cell property of Grid object

Each cell element is an integer scalar

Determines how the text is aligned within the cell.

The value is the sum of two numbers. The horizontal alignment is 1 for left, 2 for right, or for 4 center. The vertical alignment is 8 for top, 16 for bottom, or 32 for center.

The default value depends on the type of the cell. Header cells default to centered in both directions ($4 + 32 = 36$). Text cells in the data area default to top, left ($1 + 8 = 9$). Numeric cells default to top, right ($2 + 8 = 10$).

The 'tickinterval' property

Not implemented under MacOS

Integer scalar

Valid for: Trackbar

The `tickinterval` property determines the interval between the tick marks on a Trackbar control. The default is 10, corresponding to 10 intervals between the default minimum and maximum values of 0 and 100 (as set by the `range` property).

If you set this value to 0, the ticks will not be displayed (unless you set them explicitly using the `ticks` property).

Under MacOS, this property is not implemented. The control will always display with a fixed number of tick marks.

The 'tickpos' property

Implemented under Windows only

Read-only, Two-column numeric matrix

Valid for: Trackbar

The `tickpos` property returns a two column matrix, with one row for each tick mark in the control. The first column gives the logical value associated with each tick, the second column gives the physical position of the tick marks within the client area of the control, in the current `scale` of the object. For vertical trackbars, the vertical position of the ticks is returned, and for horizontal trackbars, the horizontal position of the ticks is returned.

The 'ticks' property

Implemented under Windows only

Integer vector

Valid for: Trackbar

Usually, a Trackbar control includes tick marks at regular intervals set by the `tickinterval` property. If you want to show tick marks at a different set of values (for example, to indicate a logarithmic scale), you can do so by assigning a vector of integers to the `ticks` property. These values should be within the minimum and maximum values set by the `range` property.

The 'tie' property

An integer scalar

Valid for: All objects

The `tie` property is mainly used for compatibility with the older APL.68000 method of creating windows using the `NEWWINDOW` function, where windows are identified by tie numbers. However, it can also be used to hold any integer scalar value you choose, and is available in event records returned by `DEV`. If you use the `NEWWINDOW` function to create a window, the `tie` property will be set to the tie

number passed as the first element of the right argument of `NEWWINDOW`. Objects created with `NEW` or `WI` will be assigned a unique default tie value by the system, using the following convention:

10 - 99 Window, Form or Dialog

10000+ Control created using `NEW` or `WI`

Thus, if you wish to use the tie property to identify objects, and you are also using the `NEWWINDOW` function, it is a good idea to use the range 100 to 9999 for your own tie numbers to ensure uniqueness, but you are not compelled to do so.

The 'timeout' property

Integer Scalar

Valid for: `GetMail`, `SendMail`, `HTTPClient`, `Socket`

The `timeout` property sets the timeout (in milliseconds) before the communication will be abandoned if no reply is received. A value of `-1` means 'wait for ever'.

The 'title' property

An alternative name for the `caption` property.

The 'to' property

Character Vector

Valid for: `GetMail`, `SendMail`

The `to` property contains a list of recipients of an e-mail. The recipients are specified as a comma-delimited list of e-mail addresses.

For the `SendMail` object, you should set this property before calling the `SendMessage` method. The default is an empty vector.

For the `GetMail` object, this property is read-only, and is valid only after you have called the `GetMessage` method to retrieve an e-mail from the POP3 server.

The 'tooltip' property

A character vector

Valid for: Most visible controls

A character vector which gets displayed as help text when the user moves the mouse pointer over the control and pauses. You normally use this to provide information about what the control does. (It will be suppressed if you set the `tooltipenabled` property of the system object to 0).

The 'tooltipenabled' property

An boolean scalar

Valid for: System object

This property determines whether tooltips are displayed in the application. If it is set to 1 (the default), the `tooltip` property (if set) of a control will be used to display help text when the mouse hovers over the control. If it is set to 0, tooltips are suppressed.

The 'type' property

Character Vector (For Chart and Series), Integer Scalar (for Socket)

Valid for: Chart, Series, Socket

Chart Object

The `type` property for a Chart object determines the type of graph which will be drawn. It is one of: 'line' 'stair' 'area' 'scatter' 'bar' 'stackedbar' 'horizbar' 'horizstackedbar' 'hilo' 'candle' 'pie' or 'mixed'. See the separate section on properties of the Chart object for details.

Series Object

Where a Chart object is of type 'mixed', each Series displayed on the chart will be displayed according to its own `type` property. The valid values are 'line' 'stair' 'area' 'scatter' 'bar' 'hilo' and 'candle'

Socket Object

For a Socket object, the `type` property is an integer which determines the type of socket connection required. The valid values are: 1 = Stream socket, 2 = Datagram socket, 3 = Raw-protocol interface, 4 = Reliably-delivered message, 5= Sequenced packet stream.

The default is 1.

The 'unicode' property

Integer vector (can also be written as a character vector)

Valid for: System object

Unicode is a worldwide standard which encodes characters in two bytes, thus providing a total of 65,536 possible characters. This is enough to represent all the international and special characters used in modern computer applications. APL characters, including all those used in APLX, are defined in the Unicode standard.

The `unicode` property of the System object allows your program to exchange Unicode text with other applications via the clipboard. When you read the property, any Unicode text in the clipboard is returned as a vector of integers. (You can translate it to the APLX internal text representation by using `⍲UCS`). You can write either a vector of integers, each representing a valid Unicode character in the range 0 to 65535, or a character vector. If you write a character vector to this property, it is translated to Unicode and placed on the clipboard.

The 'units' property

Read-only: Numeric 6 by 2 matrix

Valid for: All displayable objects, and the Printer and System objects

The `units` property returns the number of pixels per unit, for each of the possible values of the `scale` property. The first column refers to the vertical axis, and the second refers to the horizontal axis. Each

row refers to one of the possible scale values (1 = character units, 2 = dialog units, 3 = points, 4 = twips, 5 = pixels, 6 = millimetres). The values depend on the resolution of the output device.

For example:

```

      '□' DWI 'units'
16      8
2      2
1.333333333 1.333333333
0.06666666667 0.06666666667
1      1
3.779527559 3.779527559

```

This means that, for character units on the current device (the screen), there are 16 pixels per unit in the vertical direction and 8 in the horizontal direction.

The 'update' property

Boolean Scalar

Valid for: Chart, Picture

Normally, when you change a property or invoke a method which has a visual effect, the screen is immediately updated to reflect the change. For example, if you change the title of a Chart, it is immediately redrawn. Sometimes, however, you may wish to suppress re-drawing temporarily - for example, if you are making lots of changes to a Chart, or to an Image object which is displayed in a Picture, and you do not want to redraw it until all the changes have been made. The `update` property can be used to suppress and enable redraws.

The default value for `update` is 1, meaning that updates take place immediately (when events are next handled). If you set it to 0, no drawing or re-drawing will take place. When you set it to 1 again, the whole control will be redrawn, and from then on updates will be processed as normal.

Changing the `update` property should be done with care, since if you set it to 0 and do not set it back to 1, the control will not respond to update events and will not display correctly.

The 'url' property

Read-only, Character Vector

Valid for: Browser, HTTPClient

The `url` property can be read after you have successfully loaded a web-page in a Browser or HTTPClient object. It contains the actual Uniform Resource Location (i.e. web page address) which has been retrieved. This may not be the same as the page you requested, because the request may have been re-directed.

The 'usealtscale' property

Boolean Scalar

Valid for: Series

This property is a Boolean scalar. It defaults to 0, meaning that the Series should be drawn using the main Y scale of the parent Chart object (i.e. the left axis). If you set it to 1, the Series will instead be drawn using the alternate Y scale (i.e. the right axis).

The 'user' property

Character Vector

Valid for: GetMail, SendMail, HTTPClient

Before you can use the GetMail or SendMail objects to retrieve or send mail, you need to call the `Open` method to connect to the mail server which you specify using the `host` property. Depending on how the server is set up, you will probably need to set a user name and password before calling `Open`. The user name will normally be supplied by your Internet Service Provider (ISP) or network administrator.

For an example, see the description of the `host` property.

For the HTTPClient object, the `user` property should be set to access pages which require a user name and password.

The 'valid' property

Read-only Cell property of Grid object

Each cell element is a Boolean scalar

Returns 1 if the cell is valid, and 0 if it is invalid. Text cells are always valid. Numeric cells may be invalid if the user has edited the cell contents.

The 'value' property

0 or 1 for Check, Radio ToolButton or Menu

An integer scalar or vector for a List or Combo

An integer scalar for a Selector, Spinner or Trackbar

An integer vector for Scroll or Progress

An integer scalar for a Tree

For a Grid, a character vector or numeric scalar for each cell.

Valid for: List, Combo, Check, Radio, ToolButton, Menu, Scroll, Progress, Selector, Tree
Cell property for a Grid object

The `value` property contains the current selected value of the control to which it relates. For a Menu item, Check box, Radio button or ToolButton, this is 0 if the control is not selected and 1 if it is selected. If you set a Radio button's or ToolButton's `value` to 1, any other Radio button or ToolButton in the same group is switched off, and an `onClick` event is generated. Setting the `value` property of a Check box to 0 or 1 also generates an `onClick` event.

For a List or Combo, the `value` property contains the number (in index origin 1) of the selected item or items (if any). For a single-select list box, it is a vector of length 0 (if no item is selected) or 1 (if an item is selected). For a multi-select list box, it is a vector of arbitrary length. Again, setting the `value` property of a List generates an `onClick` event.

For a Spinner or Trackbar, the `value` property is an integer scalar representing the currently-selected value.

For a Selector, the `value` property holds the index (starting at 1) of the currently-displayed page, or 0 if none.

For a Tree object, the `value` property holds the ID of the currently-selected node

For a Progress or Scroll bar, the `value` property is a four- or five-element vector, as follows

- [1] Current value (initially 0)
- [2] Maximum value (default 100)
- [3] 'Page' increment (default 1)
- [4] Minimum value (default 0)
- [5] *Scroll bar only* (Reserved: 0)

When setting this property, you do not have to set all elements.

Grid object `value` cell property

When you read this cell property for a Grid object, it returns a character vector for each Text cell and a numeric scalar for each Numeric cell, for each cell in the list of rows and columns specified. If a Numeric cell is invalid, it will return the current `conversionerrorvalue`. If you read a set of cells some of which are Numeric and some Text, it will return a nested array containing a mixture of character vectors and numeric scalars. If you read a set of cells all of which are Numeric, it will return a simple numeric array.

When you write to this cell property for a Grid object, it sets the value for each cell in the list of rows and columns specified. If you supply a number for a cell, that cell becomes a Numeric cell. If you supply a character vector or scalar, it becomes a Text cell. Writing to a cell also causes its `valid` property to be set to 1.

Although `value` is a cell property for a Grid object, as a convenience you can read (but not write to) `value` as a simple property of the Grid object. It returns the value of the currently-active cell. If the cell is a text cell, it will return a character vector. If it is a numeric cell, it will return the numeric value of the cell, or (if the cell is invalid) the current `conversionerrorvalue`.

The 'values' property

Numeric vector or matrix

Valid for: Series

The `values` property provides a quick way of specifying the X and Y values of a Series in one operation. You can assign to it in the following ways (they are tested for in this order):

(a) *2 by N numeric matrix:*

In this case the first row is assumed to contain the X values, and the second row the Y values.

(b) *N by 2 numeric matrix:*

In this case the first column is assumed to contain the X values, and the second row the Y values. (Note: for a 2 by 2 matrix, case (a) applies).

(c) Numeric scalar, vector, 1 by N matrix, or N by 1 matrix;

In this case the data provided is assumed to represent the Y values of the series, and the X values are implicitly set to 0, 1, 2....N-1

If you read back this property, the data is always returned in format (a), i.e. as a 2 by N matrix of X values, Y Values

Note that you can also set the X and Y values separately using the `xvalues` and `yvalues` properties.

The 'verbs' property

Read-only, nested vector of character vectors

Valid for: OCX controls and OLEContainer (*Implemented under Windows only*)

The `verbs` property returns a nested vector of the application-specific 'verbs' which the server application defines. Each verb is a character string giving the name of an action which the control can perform (possibly with an ampersand character to indicate a menu mnemonic character).

You can use this information to instruct the external control to carry out a specific operation, using the `DoVerb` method. The parameter which you pass to `DoVerb` corresponds to the position (in index origin 1) of the name of the operation in the `verbs` property.

There are also pre-defined standard verbs identified by 0 or negative IDs. These do not appear in the `verbs` property.

See the `DoVerb` method for more information.

The 'version' property

Read-only, Integer vector

Valid for: System object

Reading the `version` property of the System object returns information about the operating system and APLX GUI sub-system. It returns a five-element integer vector containing:

- [1] Version number for the System Class sub-system
- [2] GUI environment code 0 = MacOS 1 = Windows, 2 = X-Windows, 4 = Console

The 'volume' property

Implemented under MacOS only

Scalar number in the range -1.0 to $+1.0$

Valid for: Movie

Set movie volume in range -1.0 to $+1.0$. Values between 1.0 and 0.0 cause sound to be reduced proportionately. Negative values mute the sound and can be used to stop the sound temporarily whilst retaining information about the volume previously chosen.

The 'where' property

Two- or four-element numeric vector

Valid for: All displayable objects

The `where` property determines the position of the top left corner of an object relative to its parent, in the `scale` set for the *parent* object (by default, character units). It optionally also sets the size of an object. If set to a two-element vector, the first element is the vertical position and the second the horizontal position (0,0 being the Top, Left of the parent). If set to a four-element vector, the third and fourth elements are the height and width of the object. When referenced, `where` returns a four-element vector.

The 'winptr' property

Synonym: hdc

Read-Only, Integer Scalar

Valid for: Window, Document, Dialog. Under Windows, also valid for Check, Radio, Button, Progress, Trackbar, Spinner, Scroll, List, Combo, Edit, RichEdit, Movie, Selector, Page, Treeview, Frame, Grid

The `winptr` property is used for low-level programming only. Under MacOS, its value is the `GrafPtr` pointer to window. It can be passed to MacOS calls which require a `GrafPtr` argument. You can also retrieve the `WindowPtr` as the `handle` property.

Under Windows, the `winptr` property represents the device context (HDC) of the window or control. It can be passed to Windows calls which perform graphics operations.

The 'workarea' property

Read-only, Four-element numeric vector

Valid for: System

The `workarea` property of the System object returns the rectangle of the usable screen size, in System object's current `scale` (by default, character units). The first two elements are the Top and Left of the usable area. The third and fourth elements are the Height and Width of the usable area.

The distinction between the 'usable' screen rectangle and the full screen size (as returned by the `size` property of the System object) is that the usable screen rectangle takes account of any space reserved by the operating system (for example, for the Windows 'taskbar' or the MacOS 'dock'). When creating windows, you should normally ensure that they fit within the rectangle returned by the System `workarea` property.

Under Linux, the `workarea` property may return the full size of the screen rather than the area which is truly usable by applications. This is because the usable area depends on the particular X Window manager which is in use.

The 'wrap' property

Boolean scalar

Valid for: Spinner

The `wrap` property for a Spinner object is 1 if the value wraps round to the minimum value when the maximum value is reached (and vice versa), or 0 otherwise. The default is 0.

The 'wssize' property

Integer scalar

Valid for: APL (child task) object

By default, a new child task is allocated the amount of workspace which you have set as the default using the Preferences dialog. The `wssize` property allows you to specify a different size, as a value in bytes. If you ask for a very small value, a minimum value of around 100KB will be allocated. If you ask for a very big value, the operating system may allocate a smaller value. A value of 0 means 'use the default'. You must set the `wssize` property before calling the `Open` method, but you can read it back at any time.

Note for users of APLX64: Because the System Class sub-system runs as a 32-bit application even under APLX64, special provisions apply if you are creating a 64-bit workspace bigger than 2 GB. In this case, the workspace size in bytes cannot be expressed as a 32-bit integer, so the `wssize` property becomes internally represented as a floating-point number. It still has to be a whole number, of course.

The 'xaltintercept' property

Read-only, Numeric Scalar

Valid for: Chart

Returns the X position where the alternate Y axis (if any) crosses the X axis. Because the alternate Y axis is always drawn on the right-hand edge of the chart, this property is read-only.

The 'xaxislabel' property

Character Vector

Valid for: Chart

This property allows you to specify a label for the X axis, such as 'Year' or 'Velocity (km/sec)'

The 'xclasses' property

Read-only, 4-column nested matrix of character vectors

Valid for: System object

The `xclasses` property of the System object returns a nested matrix which gives details of the OCX controls installed on your system. These are external controls which are separate from APLX, but which you can place in your APLX windows and use just as you would a built-in APLX control. (Information about OCX controls is also shown in the Control Browser on the Tools menu).

The returned array has one row for each OCX control installed. However, not all of these will necessarily be usable from APLX, since they may be private controls which are not designed for use by other applications.

The first column is the plain text name, for example 'Chart FX'. The second column is an alphanumeric string enclosed in curly brackets, which is the unique ID of the OCX control. The third column is the object class name, which is usually a period-delimited string giving the vendor name, object class name and optionally the version number, for example 'SoftwareFX.ChartFX.20'. (The fourth column is reserved for future use).

You can use any of these as the class name to identify the control to APLX, but we recommend that you use the third.

Under MacOS and Linux, this property always returns an empty matrix because OCX controls are not implemented.

Example

Retrieve the list of OCX controls installed on the system:

```
OCX←'#' □WI 'xclasses'
ρOCX
183 4
```

In this case there are a total of 183 OCX controls installed.

'Formula One' is an OCX control written by Visual Components, Inc. If it is installed on the system, it should appear in the list of classes returned by the `xclasses` property:

```

OCX[;1] = 'VCI Formula One Workbook'
8
OCX[8;]
VCI Formula One Workbook {042BADC5-5E58-11CE-B610-524153480001} VCF1.VCF1Ctrl.1.1

```

You can use any of the three strings to identify the control. The following sequences are all equivalent - they each create a window and place a Formula One control on it:

(a) Use the plain name to identify the OCX class:

```

MYWIN ← '□' □NEW 'Window'
MYWIN.F1.New 'VCI Formula One Workbook'

```

(b) Use the unique class ID:

```

MYWIN ← '□' □NEW 'Window'
MYWIN.F1.New '{042BADC5-5E58-11CE-B610-524153480001}'

```

(c) Use the object class name:

```

MYWIN ← '□' □NEW 'Window'
MYWIN.F1.New 'VCF1.VCF1Ctrl.1.1'

```

The 'xintercept' property

Numeric Scalar

Valid for: Chart

Determines the X position where the main Y axis of a Chart crosses the X axis. By default, this will be the left edge of the chart, or 0 if the X axis ranges from negative to positive values.

The 'xlabels' property

Nested Vector of Character Vectors (or simple Character Vector/Matrix - see text)

Valid for: Chart

The `xlabels` property can be used to change the labels written next to the major tick marks on the X axis. By default, the Chart object labels the major ticks with the corresponding numeric value. If you specify this property, then your labels will be used instead. However, (except for a bar chart or pie chart), you must also specify the corresponding major tick positions using the `xmajorticks` property, otherwise the Chart object will not know where the labels should be placed.

The labels can be specified either as a character vector with embedded carriage returns, or as a text matrix, or as a nested vector of character vectors. The first label is used for the first major tick position, and so on. If there are more labels specified than tick positions, the surplus labels are ignored. If there are fewer, the ticks which do not have user-defined labels are labelled with the default numeric value.

When you read this property back, APLX always returns a nest of character vectors.

The 'xlogscale' property

Boolean Scalar

Valid for: Chart

If this property is set to 0 (which is the default), a normal linear scale applies for the X axis. If set to 1, a logarithmic scale is used for the X axis.

Note: You cannot set a logarithmic scale for an axis where any data point (or intercept) is 0 or negative. If this situation occurs, the scale will revert to linear.

The 'xmajorticks' property

Numeric Vector

Valid for: Chart

This property allows you to specify the positions for the major ticks along the X axis, overriding the Chart object's automatic selection of tick positions. You can label the ticks using the `xlabels` property.

The 'xminorticks' property

Numeric Vector

Valid for: Chart

This property allows you to specify the positions for the minor ticks along the X axis, overriding the Chart object's automatic selection of tick positions.

The 'xscale' property

Read-only, Two-Element Numeric Vector

Valid for: Chart

This property returns the lower and upper values of the X scale, which will either have been chosen automatically by the Chart object based on the data, or will be determined by the lowest and highest user-defined tick positions.

The 'xvalues' property

Numeric Vector (or one-row matrix)

Valid for: Series

This property can be written as a numeric scalar, vector, or one-row matrix. It allows you to specify the X values for points on the series (see the `values` property for an alternative method). If left as an empty vector, the X values are implicitly assumed to be 0, 1, 2...

The 'yaltaxislabel' property

Character Vector

Valid for: Chart

This property allows you to specify a label for the alternate Y axis, such as 'Smokers (% of adult population)'.

The 'yaltlabels' property

Nested Vector of Character Vectors (or simple Character Vector/Matrix - see text)

Valid for: Chart

The `yaltlabels` property can be used to change the labels written next to the major tick marks on the alternate (right-hand) Y axis, if any. By default, the Chart object labels the major ticks with the corresponding numeric value. If you specify this property, then your labels will be used instead. However, you must also specify the corresponding major tick positions using the `yaltmajorticks` property, otherwise the Chart object will not know where the labels should be placed.

The labels can be specified either as a character vector with embedded carriage returns, or as a text matrix, or as a nested vector of character vectors. The first label is used for the first major tick position, and so on. If there are more labels specified than tick positions, the surplus labels are ignored. If there are fewer, the ticks which do not have user-defined labels are labelled with the default numeric value.

When you read this property back, APLX always returns a nest of character vectors.

The 'yaltlogscale' property

Boolean Scalar

Valid for: Chart

If this property is set to 0 (which is the default), a normal linear scale applies for the alternate Y axis. If set to 1, a logarithmic scale is used for the alternate Y axis.

Note: You cannot set a logarithmic scale for an axis where any data point (or intercept) is 0 or negative. If this situation occurs, the scale will revert to linear.

The 'yaltmajorticks' property

Numeric Vector

Valid for: Chart

This property allows you to specify the positions for the major ticks along the alternate Y axis (if any), overriding the Chart object's automatic selection of tick positions. You can label the ticks using the `yaltlabels` property.

The 'yaltminorticks' property

Numeric Vector

Valid for: Chart

This property allows you to specify the positions for the minor ticks along the alternate Y axis (if any), overriding the Chart object's automatic selection of tick positions.

The 'yaltscale' property

Read-only, Two-Element Numeric Vector

Valid for: Chart

This property returns the lower and upper values of the alternate Y scale (if any), which will either have been chosen automatically by the Chart object based on the data, or will be determined by the lowest and highest user-defined tick positions.

The 'yaxislabel' property

Character Vector

Valid for: Chart

This property allows you to specify a label for the main Y axis, such as 'Share Price (US \$)'

The 'yintercept' property

Numeric Scalar

Valid for: Chart

Determines the Y position where the X axis of a Chart crosses the X axis. By default, this will be the bottom edge of the chart, or 0 if the Y axis ranges from negative to positive values.

The 'ylabels' property

Nested Vector of Character Vectors (or simple Character Vector/Matrix - see text)

Valid for: Chart

The `ylabels` property can be used to change the labels written next to the major tick marks on the main Y axis. By default, the Chart object labels the major ticks with the corresponding numeric value. If you specify this property, then your labels will be used instead. However, you must also specify the corresponding major tick positions using the `ymajorticks` property, otherwise the Chart object will not know where the labels should be placed.

The labels can be specified either as a character vector with embedded carriage returns, or as a text matrix, or as a nested vector of character vectors. The first label is used for the first major tick position, and so on. If there are more labels specified than tick positions, the surplus labels are ignored. If there are fewer, the ticks which do not have user-defined labels are labelled with the default numeric value.

When you read this property back, APLX always returns a nest of character vectors.

The 'ylogscale' property

Boolean Scalar

Valid for: Chart

If this property is set to 0 (which is the default), a normal linear scale applies for the main Y axis. If set to 1, a logarithmic scale is used for the Y axis.

Note: You cannot set a logarithmic scale for an axis where any data point (or intercept) is 0 or negative. If this situation occurs, the scale will revert to linear.

The 'ymajorticks' property

Numeric Vector

Valid for: Chart

This property allows you to specify the positions for the major ticks along the Y axis, overriding the Chart object's automatic selection of tick positions. You can label the ticks using the `ylabels` property.

The 'yminorticks' property

Numeric Vector

Valid for: Chart

This property allows you to specify the positions for the minor ticks along the Y axis, overriding the Chart object's automatic selection of tick positions.

The 'yscale' property

Read-only, Two-Element Numeric Vector

Valid for: Chart

This property returns the lower and upper values of the main Y scale, which will either have been chosen automatically by the Chart object based on the data, or will be determined by the lowest and highest user-defined tick positions.

The 'yvalues' property

Numeric Vector (or one-row matrix)

Valid for: Series

This property can be written as a numeric scalar, vector, or one-row matrix. It allows you to specify the Y values for points on the series (see the values property for an alternative method).

Δ.. Delta property

Any APL array or overlay

Valid for: All objects

Any property name which starts with the delta character Δ can be used to store your own data in the object. Subject to available memory, you can have as many such properties as you like, and you can store any simple or nested array, or overlay, in the property. Referencing a delta property to which no data has been assigned generates a VALUE ERROR, just like a standard APL variable:

```

W←'□' □NEW 'Window'
W.ΔTYPE←'SCATTER'
W.ΔVALS←13.4 66.7 12.2
W.ΔTYPE
SCATTER
W.ΔVALS
13.4 66.7 12.2
W.ΔXPOS
VALUE ERROR
W.ΔXPOS
^

```

Using Delta properties to share data between tasks

Delta properties work in a special way for APL (child task) objects. A delta property for the System object of a child task is the same as the delta property of the same name in the parent task's APL (child task) object. Thus the parent task can assign arbitrary data to a delta property, and the child task can access it through its own System object. If the child task writes to a delta property of its System object, the parent can see the data which has been assigned in the child task object:

```

Task1←'□' □NEW 'APL' ♦ Task1.wssize←200000 ♦ Task1.background←1
Task1.ΔMESSAGE←'Startup'
Task1.Open
Task1.Execute "'#' □WI 'ΔMESSAGE' 'OK'"
Task1.ΔMESSAGE
OK

```

This feature can be used in a number of ways. When starting a task, it can be used to pass an overlay containing all the functions and variables which the child task will need to run. When the task is running, delta properties can be used to share state information, parameters and results between the parent and the child.

The `delta` property persists for as long as the child task object exists in the parent task. This means that a child task can write to a `delta` property, and then terminate. The data will still be available to the parent.

Special considerations for Client-Server versions of APLX

Because the whole System Class sub-system runs on the Client machine, the data contained in a `delta` property is held (in memory) on the Client machine, as a variable in APLX 32-bit format (even if the APL session which wrote the variable is a 64-bit implementation of APLX). The data must therefore be representable as a 32-bit object, which means it must be smaller than 2GB and must not contain more than 2147483647 elements. An array which contains 64-bit integer numbers of magnitude bigger than $2 \cdot 31$ will be converted to float data, and precision may be lost.

Section 4. System Classes: Methods

The 'Abort' method

Argument: None

Result: None

Valid for: Printer

Call the `Abort` method if you want to abort a current print job. It has the effect of stopping printing, throwing away any pending printer output, and closing the Printer object. It takes no arguments.

The 'Accept' method

Argument: None

Result: Integer Scalar

Valid for: Socket

This method should be called for a Socket object which is acting as a server. You first call the `Listen` method, which causes the underlying socket to be linked to a specific port. You then typically loop round calling the `Accept` method. This will cause your task to block until a remote client tries to connect to your socket (or until the `timeout` property has expired, if this is sooner). When the client connects, a new socket is created and linked to the client. The `Accept` method returns the handle of this new socket. Your APL application can then create a new Socket object, and write the returned handle to the `handle` property of the newly created object, which can then be used to communicate with the client. (You will probably want to create the new Socket object in a different APLX task, so that the main server task can loop round and wait for another client connection).

The return value from the `Accept` method is a positive integer if a client connected (this is the handle you should use for further communication). It is 0 if the method timed out or was interrupted. It is `-1` if an error occurred.

The 'Addimages' method

Argument: See below

Result: None

Valid for: ImageList (*Not implemented under MacOS*)

The `Addimages` method adds one or more images to the ImageList object. It accepts any of the following data forms:

- A character vector giving an image file name (a bitmap, cursor, or icon file).
- An integer matrix of N by M pixel `COLORVAL` color values.
- A pair of character vectors, the first giving an image file name, the second giving a transparency mask image file name.
- A pair of data items, the first a character vector giving an image file name, the second indicating which color should be regarded as transparent. The format of the second element can be either `('color' COLORVAL)` or `('color' R G B)`.
- A pair of integer matrices, the first giving pixel color values, the second giving a transparency mask.
- A pair of data items, the first an integer matrix of pixel color values, the second indicating a transparency color in the form `('color' COLORVAL)` or `('color' R G B)`.
- A nested vector of multiple instances of the above.

In the above descriptions, `R`, `G` and `B` are individual Red, Green and Blue color values, each in the range 0 to 255. `COLORVAL` is a single integer encoding the color as $256 \times \text{Blue} + \text{Green} + \text{Red}$.

A single bitmap can optionally contribute multiple images to the ImageList, laid side by side. To do this, you need first to set the `imagesize` property to be the size of the image. Any bitmap of a width N times the width specified in `imagesize` will then contribute N images to the list.

Note: The `Addimages` method and the `imagenames` property are alternative ways of setting the list of images. Images added by the `Addimages` method are not persistent if other properties (such as `imagesize`) are subsequently changed.

The 'Addrows' method

Not currently implemented

The 'Arrange' method

Not currently implemented

The 'Back' method

Argument: None

Result: None

Valid for: Browser

This method causes the Browser to go back to the previously-displayed web page. (You can tell whether this is valid by reading the `status` property).

The 'Beginlabeledit' method

Implemented under Windows only

Argument: Integer scalar

Result: None

Valid for: Tree

The `Beginlabeledit` method begins user-editing of the label of a node in a `Tree` object. The argument is the node ID. The effect is to move the focus to the label so that the user can start editing the text.

For this to be possible, you must have set the `style` property of the `Tree` to allow editing (value 16).

The 'Cancellabeledit' method

Implemented under Windows only

Argument: Boolean scalar

Result: None

Valid for: Tree

The `Cancellabeledit` method ends editing of the label of a tree item. It takes a single boolean argument: 0 = keep changes so far, 1 = revert to text before edit

The 'Chartaltpoint' method

Argument: 2-element Numeric Vector

Result: 2-element Numeric Vector

Valid for: Chart

This method converts from Chart coordinates (using the Alternate Y scale) to coordinates of the visible control on the screen. It can be useful if you are adding your own annotations or other graphic elements using the `Draw` method, and want to relate these to points of a `Series` which uses the Alternate Y Scale.

The argument is a two-element vector of Y value (using the Alternate Y scale) and X value. The result is a two-element vector of the corresponding vertical and horizontal position within the visible control (remembering that top, left is 0, 0). The result is expressed in the current `scale` of the control; you will probably want to set the `scale` property to 5 for pixels before calling this method.

The 'Chartoline' method

Argument: Integer scalar

Result: Integer scalar

Valid for: RichEdit

The `Chartoline` method allows you to determine the line number corresponding to a particular character in a RichEdit control. The argument is the character position (in index origin 0) of the character, relative to the start of the text in the control. The result is corresponding line number (also in index origin 0).

The 'Charttopoint' method

Argument: 2-element Numeric Vector

Result: 2-element Numeric Vector

Valid for: Chart

This method converts from Chart coordinates (using the main Y scale) to coordinates of the visible control on the screen. It can be useful if you are adding your own annotations or other graphic elements using the `Draw` method, and want to relate these to points of a Series which uses the main Y Scale.

The argument is a two-element vector of Y value (using the main Y scale) and X value. The result is a two-element vector of the corresponding vertical and horizontal position within the visible control (remembering that top, left is 0, 0). The result is expressed in the current `scale` of the control; you will probably want to set the `scale` property to 5 for pixels before calling this method.

The 'Clear' method

Argument: None

Result: None

Valid for: Edit, RichEdit, Document, SendMail

For the Edit, RichEdit, and Document classes, the `clear` method deletes the currently selected text (if any) from the object. If there is no text currently selected, it does nothing. This is what normally should happen when Clear is selected from the Edit menu. The `onChange` and `onClick` callbacks (if any) are triggered if text was deleted.

For the SendMail class, the `clear` method clears all the properties associated with a specific e-mail (subject, body, attachments, and so on). If you are sending a series of e-mails, you should normally call this method in between each one, in order to ensure that you do not accidentally send the wrong information to the next recipient.

The 'Click' method

Argument: None

Result: None

Valid for: Any visible object, but usually applies only to Buttons

Invoking the `click` method for an object invokes its `onClick` callback (if any). It is normally used for triggering the same action as would be invoked if the user clicked in a button. It is useful for avoiding duplication of code in cases where some other part of your program needs to be able to carry out the same action as happens when a button is clicked.

The 'Clienttoscreen' method

Argument: 2-element integer vector

Result: 2-element integer vector

Valid for: Any visible control

This method converts coordinates within the client area of a control to coordinates relative to the whole screen. Both the argument and the result are two-element vectors of Row position, Column position (i.e. Y, X), always expressed in pixels. The `scale` properties of the control and System objects are ignored.

The 'Close' method

Argument: Optional boolean scalar - see text

Result: None

Valid for: Any object except System, but usually applies only to Form, Window, Document, Dialog, APL, SendMail, GetMail, Socket, and Printer objects

The `Close` method is used to request an object to close itself.

Window-level and other visible objects

For a Window, Form, Dialog or Document, and for other visible objects, the exact behavior depends on the event handler state and the argument to the `Close` method.

- If the `Close` method is invoked with an argument of 1, the object is closed unconditionally (but not destroyed), and the `onClose` callback (if any) is not invoked.
- If the `Close` method is invoked without an argument (or with an argument of 0), and if an `onClose` callback has been defined for the object, the `onClose` callback will be triggered. This callback function - which you supply - will normally do any processing necessary when the object is closed (such as saving the state of the window), and then actually close the object by invoking the `Close` method again with an argument of 1, or alternatively destroy it altogether using the `Delete` method. In some cases, however, the object may 'refuse' to close itself; for example, the `onClose` callback function might put up a dialog box to ask the user to confirm that he wishes to close the window, and if not then the window is not closed.
- If no `onClose` callback has been defined for an object, the default behavior is to close the object immediately, whether or not an argument was supplied to the `Close` method.
- If the object is already in the closed state the `Close` method has no effect at all.

When an object is closed it is no longer visible, but it still exists. You can still read or set its properties, and it can subsequently be re-opened in the same state by using the `Open` method. However, you should remember that a closed object continues to use system memory, and therefore if an object is not going to be needed again you should destroy it altogether (using the `Delete` method) as soon as you can, for example at the end of the `onClose` callback function.

For a `Form`, `Dialog`, `Window` or `Document` object, invoking the `Close` method without an argument has the same effect as the user clicking in the close box in the window's title bar.

Printer object

For a `Printer` object, the `Close` method has a special effect. The current print job is closed, and submitted to the printer. The method takes no argument.

APL (child task) object

For an `APL` object, the `Close` method terminates the child task, closing any windows belonging to the task and freeing the workspace memory. The method takes no argument.

Networking objects

For the `SendMail`, `GetMail` and `Socket` objects, the `Close` method closes the connection and releases resources at the other end. It takes no arguments. You should always ensure you call the `Close` method when the communications session is complete.

The 'Closedocument' method

Argument: None

Result: None

Valid for: `OLEContainer` (*Implemented under Windows only*)

The `Closedocument` method closes the document in an `OLEContainer`, remembers its state, and disconnects from the server application. You can still read the `contents` property to retrieve the document contents.

The 'Copy' method

Argument: None

Result: None

Valid for: Edit, RichEdit, Document, Browser, Chart, Image

For an Edit, RichEdit, Document, or Browser control, the `COPY` method copies the currently selected text (if any) to the clipboard. If there is no text currently selected, it does nothing. This is what normally should happen when Copy is selected from the Edit menu. (For a RichEdit control, the text written to the clipboard includes formatting information such as the font and text color.)

For a Chart or Image object, the `COPY` method copies the current image to the clipboard, as a resizable picture.

The 'Create' method

Argument: See text

Result: None

Valid for: Any class of object except the System object

The `CREATE` method is identical to the `NEW` method, except that if an object of the same name already exists, it is automatically deleted before the new object is created.

The 'Cut' method

Argument: None

Result: None

Valid for: Edit, RichEdit, Document

The `CUT` method copies the currently selected text (if any) to the clipboard, and then deletes the selected text from the object. If there is no text currently selected, it does nothing. This is what normally should happen when Cut is selected from the Edit menu. The `onChange` callback (if any) is triggered.

For a RichEdit control, the text written to the clipboard includes formatting information, such as the font and color.

The 'Delete' method

Argument: None

Result: None

Valid for: Any object except the System object, but usually applied to a Window, Document or Dialog.

The `Delete` method destroys an object and releases all the memory it uses. After you have deleted an object, it can no longer be re-opened and you can no longer read or set its properties. If the object contains other objects, they are also destroyed, so normally you do not need explicitly to destroy child objects. You should always delete a window which will no longer be needed, since otherwise the object continues to use system memory (until APLX terminates execution, when all objects are deleted automatically). If you created it using `NEW`, it will automatically be deleted when the last reference to it is erased from the workspace.

When an object is deleted, the `onDestroy` callback of the object is invoked,

The 'Deletecols' method

Argument: Integer vector

Result: None

Valid for: Grid

The `Deletecols` method takes an argument which is a vector of columns to be deleted from a Grid object. After they have been deleted, the remaining columns are re-numbered to form an integer sequence starting at 1.

The 'Deletemessage' method

Argument: Integer Scalar

Result: None

Valid for: GetMail

This method can be called to mark a message on the POP3 server for deletion. It takes a single argument, which is the index of the message to be deleted, in the range 1 to N, where N is the number of messages on the server (as returned by the `count` property, or the number of rows in the `messages` property).

The server will not actually delete the message until the session is successfully completed and you call the `Close` method. If an error occurs, or if the connection is lost, the message will be left untouched.

See also the `deleteonread` property, which allows you to specify that messages should automatically be marked for deletion when they are downloaded.

The 'Deletenodes' method

Argument: Integer vector

Result: None

Valid for: Tree

The `Deletenodes` method deletes one or more nodes from a `Tree` object. The argument is an integer vector of the node IDs to be deleted.

The 'Deleterows' method

Argument: Integer vector

Result: None

Valid for: Grid

The `DeleteRows` method takes an argument which is a vector of rows to be deleted from a Grid object. After they have been deleted, the remaining rows are re-numbered to form an integer sequence starting at 1.

The 'Doverb' method

Argument: Integer scalar

Result: None

Valid for: OCX controls and OLEContainer (*Implemented under Windows only*)

The `DoVerb` method takes an integer argument which is the ID of a 'verb' to perform. This can either be a positive integer representing one of the application-specific verbs (see the `verbs` property), or one of the following standard verbs:

- 0 Specifies the action that occurs when an end-user double-clicks the object in its container.
- 1 Instructs an object to show itself for editing or viewing.
- 2 Instructs an object to open itself for editing in a window separate from that of its container.
- 3 Causes an object to remove its user interface from the view. Applies only to objects that are activated in-place.
- 4 Activates an object in place, along with its full set of user-interface tools, including menus. If the object does not support in-place activation, gives domain error
- 5 Activates an object in place without displaying tools, such as menus and toolbars, that end-users need to change the behavior or appearance of the object.
- 6 Used to tell objects to discard any undo state that they may be maintaining without deactivating the object.

The 'Draw' method

Argument: Nested array

Result: Depends on operation

Valid for: All visible controls and the Image and Printer objects

The `Draw` method allows you to draw text, lines, patterns, pictures and geometric shapes on windows, controls, charts, images and printer pages. It takes an argument which comprises one or more phrases which each start with a keyword indicating the operation to perform, followed by the arguments to that operation.

As well as drawing on the screen or printer, it can also be used to draw on an off-screen bitmap using an Image object. This can then be saved in many different formats, such as JPEG, GIF, PDF, BMP, and PNG. In addition, the `Draw` method can be used in conjunction with the Chart object to add extra graphics elements to charts.

Drawing takes place using a current Pen (for foreground drawing) and Brush (for background drawing). You can also set the Mode which determines how drawing interacts with what is already on the screen.

APLX automatically handles window refreshing for you if the window is uncovered or resized (you can switch this off using the `autoredraw` property). It does this by storing the sequence of `Draw` commands applicable to the window or control, and replaying them when an update is required. You can also group a series of commands together, and selectively enable or disable them (for example, to temporarily hide the labels on a graph), or delete them altogether from the saved sequence. This is useful for animation effects.

The general syntax of the `Draw` method is:

```
Control.Draw '<Keyword>' Arg1 Arg2...
```

where *Control* is usually the Window into which you want to draw, or a Printer, Image, Chart, Frame or Picture control.

You can also supply multiple sequences of commands on one line:

```
Control.Draw ('<Keyword1>' Arg1) ('<Keyword2>' Arg1 Arg2)...
```

See the separate section on Using the `Draw` method for full details.

The 'Editstart' method

Argument: None

Result: None

Valid for: Grid

The `Editstart` method takes no arguments. It initiates an editing session on the currently-active cell. It is useful mainly if you have set the `autoeditstart` property to 0. For example, you might initiate an edit session for certain cells only (in response to an `onSelChange` event).

The 'Eject' method

Argument: None

Result: None

Valid for: Printer

Call the `Eject` method if you want to start a new page in the current print job.

The 'Ensurevisible' method

Argument: Integer scalar

Result: None

Valid for: Tree

The `Ensurevisible` method ensures that a given node is visible, expanding and/or scrolling the tree if necessary. The argument is the node ID to be shown.

The 'Execute' method

Argument: Character vector

Result: None

Valid for: APL (child task) object

The `Execute` method allows the parent to cause the child task to execute an APL expression or system command. It takes as an argument a character string which is the command to be executed (or passed as input to `⍷` or `⍷`).

The child task should be awaiting input (you can tell whether this is the case by reading the `status` property, or by using the `onReady` callback). If it is not, the command will be placed in a buffer and will be executed when the child task next asks for input. However, there is no queue of commands held; any existing command already in the buffer will be over-written if the child task has not yet executed it. The `Execute` method returns immediately; it does not wait for the command to complete.

In this example, the child task is initially ready for input. It then executes the `)LOAD` command, and is busy for a short time (`status` property is 0). The `status` property then changes back to 1 when it is ready for further input:

```

1      ChildTask.status
ChildTask.Execute ')LOAD 10 SAMPLEEXCEL'
0      ChildTask.status
1      ChildTask.status

```

Note that in a real multi-tasking APLX application you would typically use the `Signal` method and `onSignal` events to allow the parent and child tasks to communicate with each other and synchronize their operations. The `Execute` method would typically be used only when starting up the child task.

The 'Expand' method

Argument: Two-element nested vector

Result: None

Valid for: Tree

The `Expand` method is used to expand or collapse a given node. The argument is a two element nested vector, comprising a character vector which indicates the action to be taken, and the node ID. The supported actions are:

<code>'expand'</code>	Open node to display children
<code>'collapse'</code>	Close node to hide children
<code>'toggle'</code>	Expand if closed or Collapse if open
<code>'expandbranch'</code>	Open node to display children, recursively
<code>'collapsebranch'</code>	Close node to hide children (recursively)
<code>'collapsereset'</code>	Remove all children (not implemented under MacOS)

The 'Find' method

Not currently implemented

The 'Findnode' method

Argument: Character vector or two-element nested vector

Result: Integer scalar or vector vector

Valid for: Tree

The `Findnode` method return the node identifier (or a vector of IDs) for nodes matching the specified characteristic. The first argument is a character vector giving the operation name. Some forms require a second (integer) argument, the node ID. The result is usually an integer scalar node ID, or 0 if none is found. For the `'children'` operation, a vector of IDs is returned.

Operations which do not require a second argument are:

<code>'root'</code>	The first top-level node
<code>'firstvisible'</code>	The first visible node
<code>'drop'</code>	The current drag-and-drop target
<code>'select'</code>	The currently-selected node

Operations which do require a second argument, a node ID, are:

'children'	All children of specified node (a vector is returned)
'parent'	Parent of specified node
'child'	First child of specified node
'next'	Next sibling of specified node
'prev'	Previous sibling of specified node
'nextvisible'	Next visible node after the specified node
'prevvisible'	Previous visible node before the specified node

The 'Focus' method

Argument: None

Result: None

Valid for: Window object (Window, Form, Document, Dialog) and controls which can receive the input focus. Under Windows, these are the Check, Radio, Button, Progress, Trackbar, Spinner, Scroll, List, Combo, Edit, RichEdit, Movie, OLEContainer, Splitter, Selector, Page, Tree, and Frame controls. Under MacOS, these are the List, Edit, RichEdit and Tree controls.

The `Focus` method sets the input focus to an object under program control. In the case of a top-level object, this means activating the window. In the case of a control such as an Edit object, this means the input cursor will be placed in the control (provided the window on which the control is placed is itself activated).

The 'Forward' method

Argument: None

Result: None

Valid for: Browser

This method causes the Browser to go forward one page in the history of previously-displayed web pages. (You can tell whether this is valid by reading the `status` property).

The 'Get' method

Argument: Character Vector

Result: 3-element Nested Vector

Valid for: HTTPClient

The `Get` method retrieves a page or other file from a web-server. The argument is a character vector which is the URL to retrieve (e.g. `'http://www.microapl.co.uk/apl/index.html'`).

The result is a three element vector.

The first element is the HTTP return code, which is an integer scalar. A number in the range 200 to 299 indicates success. A number in the range 400 to 499 indicates an error, such as 404 meaning 'page not found'. (See the documentation on the HTTPClient object for a list of codes)

The second element is a character vector indicating the MIME type of the returned data, such as `'text/plain'`.

The third element is a character vector containing the data - this will usually be the HTML text of the web page (possibly with embedded carriage returns). If the MIME type begins with 'text', the data will have been translated to the APLX character set.

For example:

```
H←'⍴' ⍵NEW 'HTTPClient'
R←H.Get 'http://www.microapl.co.uk/apl/index.html'
1→R
200
2→R
text/html
3→R
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<!-- $$PAGE_TITLE$$='APLX<sup><small>TM</small></sup> The exciting cross-platform
APL' -->
<!-- $$SITE_ROOT$$='..' -->

<HTML>
<HEAD>
  <TITLE>APLX: The exciting cross-platform APL</TITLE>
... etc
```

The 'Getinfo' method

Argument: Two- or three-element nested vector

Result: See text

Valid for: Tree

The `Getinfo` method returns information about one or more nodes in the tree.

The first argument is a character vector giving the attribute name you are interested in, one of: 'forceparent' 'highlightbold' 'highlightcut' 'highlightdrop' 'highlightselect' 'expanded' 'expandedonce' 'haschildren' 'wherelabel' 'whererow' 'list' 'label' 'images' or 'handle'.

The second argument is a node ID or vector of node ids to query.

The optional third argument specifies the depth of children to query. 0 means return information about the node itself only. 1 means return information about the node and its immediate children, 2 means return information about the node, its children, and grandchildren, and so on. You can also specify `-1` to indicate that information about all descendants (to any depth) should be returned.

The result is a vector or matrix (depending on attribute), with one item/row for each node. If the optional third argument is given, the result is a matrix with the first column giving node IDs, the second column giving relative depth, and remaining column(s) giving attribute values. The returned values are as follows:

Querying the state of the node(s)

Keywords: 'forceparent' 'highlightbold' 'highlightcut' 'highlightdrop' 'highlightselect' 'expanded' 'expandedonce' 'haschildren'

These each return a single boolean value for each node, indicating whether the specified attribute is true or false for each node.

Querying the labels associated with the node(s)

Keyword: 'label'

This keyword returns a character vector for each node, containing the label.

Querying the images associated with the node(s)

Keyword: 'images'

In this case a matrix is returned, with one row for each node. The matrix has four columns, containing the indices in the Tree object's `imagelist` of the non-selected image, selected image, overlay image, and user-state image. In each case, 0 is returned if there is no corresponding image associated with the node. (If the optional third argument was supplied, a 6-column matrix is returned. The first two columns are the node ID and depth, and the subsequent columns are the image indices).

Querying all the data used to set up the node(s)

Keyword: 'list'

In this case a matrix is returned, with one row for each node. The matrix has three columns, containing values of the third, fourth and fifth columns of the `list` property corresponding to the node. These are the label, the image indices vector, and the state keywords. (If the optional third argument was supplied, a 5-column matrix is returned. The first two columns are the node ID and depth, and the subsequent columns are the above data).

Querying the position and size of the node(s)

Keywords: 'wherelabel' or 'whererow'

In these cases a matrix is returned, with one row for each node. The matrix has four columns, corresponding to the top, left, height and width of the rectangle occupied by the whole node (for 'whererow') or just its label (for 'wherelabel'). The values are expressed in the current `scale` of the Tree object. Coordinates of all 0s are returned if the node is not visible because it is not expanded. Note that the coordinates may be negative or greater than the client area of the Tree control if the node is not currently scrolled into view. (If the optional third argument was supplied, a 6-column matrix is returned. The first two columns are the node ID and depth, and the subsequent columns are the top, left, height, width values).

Retrieving the internal handle of the node(s)

Keyword: 'handle'

Under Windows, this keyword returns an integer value for each node, containing a handle of type `HTreeItem` which Windows uses to identify the node internally. It can be used for low-level programming. Under MacOS and Linux, this keyword returns 0 for each node.

The 'Getmessage' method

Argument: Integer Scalar

Result: None

Valid for: GetMail

Once you have connected to a mail server, you can retrieve the full contents of an individual e-mail message (into properties of the GetMail object) by calling the `GetMessage` method. It takes an integer scalar argument, which is the index (in origin 1) of the message you want to retrieve. (This will be a number in the range 1 to the value of the `count` property, or the number of rows in the `messages` property). If successful, it will also retrieve any attachments and write them to files, in the directory specified by the `path` property.

Having successfully called `GetMessage`, you can then access various properties (such as `subject` `to` `from` `date` `body` and `attachments`) to access the content of the retrieved e-mail message.

The 'Getsummary' method

Argument: Integer Scalar

Result: None

Valid for: GetMail

Once you have connected to a mail server, you can retrieve partial details of an individual e-mail message (into properties of the GetMail object) by calling the `GetSummary` method. This takes an integer scalar argument, which is the index (in origin 1) of the message you want to retrieve. (This will be a number in the range 1 to the value of the `count` property, or the number of rows in the `messages` property).

Having successfully called `GetSummary`, you can then read the following properties for information about the e-mail: `subject` `from` `id` `header` and `size`.

The 'Head' method

Argument: Character Vector

Result: 3-element Nested Vector

Valid for: HTTPClient

The `Head` method retrieves information about a page or other file from a web-server. The argument is a character vector which is the URL (e.g. `'http://www.microapl.co.uk/apl/index.html'`).

The result is a three element vector.

The first element is the HTTP return code, which is an integer scalar. A number in the range 200 to 299 indicates success. A number in the range 400 to 499 indicates an error, such as 404 meaning 'page not found'.

The second element is a character vector indicating the MIME type of the returned data, such as `'text/plain'`.

The third element is a character vector containing the HTTP header associated with the page.

For example:

```
(errcode mimetype text) ← HTTP.Head 'http://www.microapl.co.uk/apl'
  errcode
200
  mimetype
text/html
  text
Server: Zeus/4.2
Date: Wed, 09 Feb 2005 12:19:41 GMT
Content-Type: text/html
Content-Length: 10437
Accept-Ranges: bytes
Last-Modified: Thu, 07 Oct 2004 11:35:34 GMT
```

The 'Hide' method

Argument: None

Result: None

Valid for: Any displayable object

Calling the `Hide` method is equivalent to setting the object's `visible` property to 0; it removes the object from the screen. You can make the object visible again by calling the `Show` method, or by setting the `visible` property to 1.

The 'Hittest' method

Implemented under Windows only

Argument: Two-element numeric vector

Result: Two-element nested vector

Valid for: Tree

The `Hittest` method takes a coordinate of a point in the `Tree` object, and determines whether the point is on a node and if so which node. It also returns information about which area of the node it corresponds to.

The argument is a two-element vector containing the vertical and horizontal coordinates of the point, relative to the top left of the control and expressed in the object's current `scale`.

The result is a two-element nested vector. The first element is an integer, which is the ID of the node at the given point, or 0 if the point does not correspond to a node. The second element is a character vector, indicating which area of the node the point corresponds to. It is one of the strings: 'image', 'button', 'label', 'imageuser', 'indent', 'noderight', 'above', 'below', 'toleft', 'toright', 'aboveleft', 'aboveright', 'belowleft', 'belowright' or 'nowhere', or an empty vector if the position is unknown.

The 'Insertcols' method

Argument: Integer vector

Result: None

Valid for: Grid

The `Insertcols` method takes an argument which is a vector of positions at which you wish to insert new columns in a Grid object. You can specify integers (for example 5 means 'insert before the current column 5'), or non-integers (for example 4.5 means 'insert between the current columns 4 and 5'). After the insertion, the columns are re-numbered to form an integer sequence starting at 1. Thus, if you insert at 5 and 6, the new columns become numbers 5 and 7. To insert three new columns before the current second one, specify 2 2 2.

The 'Insertnodes' method

Argument: 3-element nested vector

Result: None

Valid for: Tree

The `Insertnodes` method inserts one or more new nodes into a Tree object.

The argument is a nested vector of 3 elements. The first element is an integer scalar, representing the node ID into which the new node(s) should be placed. (You can alternatively specify the keyword 'root' to indicate that the new node(s) should appear at the top level). The second element is an integer scalar which indicates the node after which the new nodes should be placed. (You can alternatively specify a character vector, being one of 'first' 'last' or 'sort'). The third element is a 3, 4 or 5 column matrix (or vector if just one node is being added), with the same structure as for the `list` property. Each row has fields: id depth label [image_indices [state]]

The 'Insertrows' method

Argument: Integer vector

Result: None

Valid for: Grid

The `Insertrows` method takes an argument which is a vector of positions at which you wish to insert new rows in a Grid object. You can specify integers (for example 5 means 'insert before the current row 5'), or non-integers (for example 4.5 means 'insert between the current rows 4 and 5'). After the insertion, the rows are re-numbered to form an integer sequence starting at 1. Thus, if you insert at 5 and 6, the new rows become numbers 5 and 7. To insert three new rows before the current second one, specify 2 2 2.

The 'Interrupt' method

Argument: None

Result: None

Valid for: APL (child task) object

If a child task is executing (`status` property is 0), the `Interrupt` method can be called to interrupt it. It is equivalent to a keyboard interrupt (break).

The 'Job' method

Argument: None

Result: Boolean scalar

Valid for: Printer

The `Job` method displays the standard print job dialog, which allows the user to specify print parameters such as the number of copies and the collation sequence. (This is the dialog which is normally displayed when the user selects Print.. from the File menu of a typical application). It updates the current print record of the printer object, and should be called before calling the `Open` method which starts the print job.

The return value indicates how the user ended the dialog. If the user pressed the OK button, it returns 1. A return value of 0 indicates that the Cancel button was pressed, so printing should not take place.

See also the `Setup` method which displays the standard dialog for selecting document-specific parameters such as the paper size, paper source, and orientation.

The 'Linelenh' method

Argument: Integer scalar
Result: Integer scalar

Valid for: RichEdit

The `Linelenh` method allows you to determine the length of a given line of the text in the control. The argument is the line number (in index origin 0). The result is length of the line in characters.

The 'Linetochar' method

Argument: Integer scalar
Result: Integer scalar

Valid for: RichEdit

The `Linetochar` method allows you to determine the position within the text corresponding to the start of a particular line. The argument is the line number (in index origin 0). The result is the position (also in index origin 0) of the first character of the line, relative to the start of the text in the control.

The 'Listen' method

Argument: None
Result: None

Valid for: Socket

The `Listen` method is used for sockets at the server end of a network connection. It sets up the socket so that it is associated with the port specified in the `port` property.

See the description of the Socket class for details.

The 'Load' method

Argument: Character vector

Result: None

Valid for: RichEdit, Image, Browser

RichEdit

The `Load` method of a RichEdit control causes APLX to load the contents of a text file into the control (any existing contents are overwritten). The argument is the full pathname of the file you wish to load. This can be either a plain text file, or (under Windows) a Rich Text (.rtf) file, with embedded formatting information. If you specify an empty vector as the file name, a dialog is displayed to allow the user to select a file.

Image

For an Image control, the `Load` method causes APLX to load the image (or sequence of images) from the specified file into the Image. The file format can be any of those supported by ImageMagick. If you specify an empty-vector argument, a dialog is displayed to allow the user to select a file.

Browser

For the Browser object, the `Load` method takes as argument a URL. The Browser will start loading the requested page from the internet or local file.

The 'New' method

Argument: See text

Result: None

Valid for: Any class of object except the System object

Using the `new` method with dot notation

The `New` method is used to create new objects which are children of existing objects. It is thus special in that it can be invoked on an object which does not yet exist. The basic syntax is:

```
ParentRef.ObjectName.New ClassName
```

where `ObjectName` is the name of the object you want to create, and `ClassName` is the class of object you want to create, as a character vector. The name of the object must not already be in use for another object, and the class must be one of the standard APLX object class names, such as 'Button' or 'Edit'. Objects are created initially with all properties having the default values for the class of object.

For example, the following sequence creates a window containing an 'OK' button:

```
W←'□' □NEW 'Window'
W.size←8 12 ♦ W.title←'Demo'
W.But.New 'Button'
W.But.where←5 9 ♦ W.But.title←'OK'
```

Once the object has been created, and any properties specified have been set, the object is usually automatically opened, except in the case of the Printer and APL (child task) classes where you need to call the `Open` method explicitly. In addition, visual objects are automatically displayed (unless you have set the `visible` property to 0), except for the pre-defined dialog classes (`ChooseFont`, `ChooseColor`, `ChooseDir`, `OpenFile`, `SaveFile` and `MsgBox`) where the dialog does not appear until you call the `Show` method. However, for a window object (`Window`, `Form`, `Document` or `Dialog`), the window is not actually opened until the first of the following occurs:

- APL desk-calculator mode is reached
- Your program checks for events using `□WE`.
- Your program creates another top-level object using the `New` method.
- Your program invokes the `Open` or `Show` method for the window.

The reason for deferring the actual display of a window in this way is so that your program can build up the objects in the window after creating it, without the user seeing objects being created, moved, resized etc as the objects are defined.

Using the `New` method with `□WI`

Using `□WI` syntax, the `New` method can be used to create either top-level or child objects. It can also optionally set one or more properties for the newly-created object, by following the `Class` name with a list of Property-Value pairs. The following sequence is equivalent to the dot-notation example above; it creates a window containing an 'OK' button:

```
'Win' □WI 'New' 'Window'
'Win' □WI 'size' 8 12
'Win' □WI 'title' 'Demo'
'Win.But' □WI 'New' 'Button'
'Win.But' □WI 'where' 5 9
'Win.But' □WI 'title' 'OK'
```

You could achieve the same result as follows:

```
'Win' □WI 'New' 'Window' ('size' 8 12) ('title' 'Demo')
'Win.But' □WI 'New' 'Button' ('where' 5 9) ('title' 'OK')
```

If an object of the same name already exists when you try to use the `New` method, a `DOMAIN ERROR` will be generated. As an alternative, you can use the `Create` method, which is identical except that any object of the same name is automatically deleted before the new object is created.

The 'Open' method

Argument: None

Result: None

Valid for: Any object except the System object

The `Open` method explicitly opens an object, which normally makes it appear on the screen (provided its `visible` property is not zero, and also its parent is open and visible). For most ordinary visual controls, you do not normally need to use the `Open` method when an object is created, because `New` automatically opens it for you. The `Open` method is most useful for re-opening an object which has been closed, or for making a top-level object appear immediately after creation if your program has further processing to do (see the description of the `New` method). If the object is already open, this method does nothing.

For certain objects, the `Open` method has a special meaning:

For a `Printer` object, it starts a new print job, and printing commands (such as the `Print` method) will be appended to the open print job until the `Close` or `Abort` method is called.

For an APL (child task) object, the `Open` method allocates memory for the workspace and starts the task running.

For the networking objects `GetMail`, `SendMail` and `Socket`, the `Open` method establishes the connection to the remote host. See the descriptions of these objects for more details.

Note: Although you can use the `Open` method for the pre-defined dialogs such as `ChooseColor` and `OpenFile`, it has no useful effect for these object types.

The 'Overlay' method

Argument: Nested or integer vector (*see text*)

Result: None

Valid for: Image

The `Overlay` method allows you to overlay one image (the foreground image) on to another (the background image). If the foreground image is transparent, the background image will be partially visible through the foreground image.

The background image should first be loaded into an Image object. You then copy the foreground image to a specified location on the background image. The foreground image usually comes from a file, but it can come from another Image object.

To load a foreground image from file, call the `Overlay` method with this syntax:

```
Image.Overlay 'file_name' Top Left <Mode>
or
'ImageName' ⍵WI 'Overlay' 'file_name' Top Left <Mode>
```

where `file_name` is the name of the file to load (one of the standard formats supported by `ImageMagick`), and `Top` and `Left` are the positions in the Image control where it should be loaded (these can be omitted, in which case 0 0 is assumed). If the file name is an empty vector, a dialog will be invoked to allow the user to select a file. The last parameter `Mode` is normally not needed - see below.

To load a foreground image from another Image object, call the `Overlay` method with this syntax:

```
Image.Overlay Handle Top Left <Mode>
or
'ImageName' ⍵WI 'Overlay' Handle Top Left <Mode>
```

where the `Handle` parameter is an integer representing the `handle` property of the source Image object. This is very useful if you want to create the foreground image dynamically, or load a non-transparent image and make it transparent (see the `Setopacity` method)

This function loads a background image from file, and places a foreground image (also from file) over it:

```
▽DEMO_Overlay;Win
[1] '⍵' ⍵wi 'scale' 5
[2] Win←'⍵' ⍵NEW 'form' ⋄ Win.title←'Transparency' ⋄ Win.where←1 1 420 700
[3] Win.Pic.new 'Picture'
[4] Win.Pic.Img.new 'Image' ⋄ Win.Pic.Img.scale←5 ⋄ Win.Pic.align←¯1
[5] Win.Pic.Img.file←'c:\pictures\background.jpg'
[6] Win.Pic.Img.Overlay 'c:\pictures\foreground.png' 200 120
[7] ↵1 ⍵WE Win
▽
```

This version of the function does the same, but uses a second Image object to hold the foreground image and transform it, before copying it into the background:

```

    ▽DEM02_Overlay;Win;Img2;handle
[1]  '□' □wi 'scale' 5
[2]  Win←'□' □NEW 'form' ♦ Win.title←'Transparency' ♦ Win.where←1 1 420 700
[3]  Win.Pic.new 'Picture'
[4]  Win.Pic.Img.new 'Image' ♦ Win.Pic.Img.scale←5 ♦ Win.Pic.align←~1
[5]  Win.Pic.Img.file←'c:\pictures\background.jpg'
[6]  ⍎ Create a second (invisible) image object:
[7]  Img2←'□' □new 'Image'
[8]  Img2.file←'c:\pictures\foreground.png'
[9]  Img2.Transform 'Flip'
[10] handle←Img2.handle
[11] Win.Pic.Img.Overlay handle 200 120
[12] -1 □WE Win
    ▽

```

The 'mode' parameter

The underlying ImageMagick call which does the copying is `MagickCompositeImage()`. If you omit the final Mode parameter, this is called with the 'compose' type set to `AtopCompositeOp`, which means that the foreground image is placed over the background image. This is the most common requirement. However, there are other possible values which you can specify as the Mode parameter. These are defined in an 'enum' called `CompositeOperator` in the ImageMagick source; unfortunately, the exact values of the mode parameter may vary from one release of ImageMagick to another, so you will need to check the ImageMagick include files (`composite.h`) to get the numeric values for the particular version you are using - they start with `UndefinedCompositeOp = 0`.

Useful values include:

`OverCompositeOp`

The result is the union of the the two image shapes with composite image obscuring image in the region of overlap. If the foreground pixel is part or all transparent, the corresponding background pixel will show through.

`InCompositeOp`

The result is simply the foreground image cut by the shape of the background. None of the image data of the background image is included in the result.

`OutCompositeOp`

The resulting image is foreground image with the shape of image cut out.

`AtopCompositeOp`

Same as `OverCompositeOp`, except the portion of the foreground image outside of the background image's shape does not appear in the result.

`XorCompositeOp`

The result is the image data from both the foreground and background images that is outside the overlap region. The overlap region will be blank.

`PlusCompositeOp`

The result is just the sum of the image data. Output values are cropped to 255 (no overflow). This operation is independent of the matte channels.

`MinusCompositeOp`

The result of foreground image - image, with overflow cropped to zero. The matte channel is ignored (set to 255, full coverage).

`AddCompositeOp`

The result of foreground image + image, with overflow wrapping around (mod 256).

`SubtractCompositeOp`

The result of foreground image - image, with underflow wrapping around (mod 256). The add and subtract operators can be used to perform reversible transformations.

`DifferenceCompositeOp`

The result of $\text{abs}(\text{foreground image} - \text{image})$. This is useful for comparing two very similar images.

`MultiplyCompositeOp`

Multiplies the color of each target image pixel by the color of the corresponding foreground image pixel. The result color is always darker.

`BumpmapCompositeOp`

The result image shaded by the foreground image.

`ReplaceCompositeOp`

The resulting image is image replaced with foreground image. Transparency is ignored.

`CopyCompositeOp`

Replace the target image with the foreground image.

`CopyRedCompositeOp`

Copy the red channel from the foreground image to the target image.

`CopyGreenCompositeOp`

Copy the green channel from the foreground image to the target image.

`CopyBlueCompositeOp`

Copy the blue channel from the foreground image to the target image.

`ClearCompositeOp`

Make the background image transparent. The contents of the foreground image are ignored.

`DissolveCompositeOp`

Use to foreground image to 'dissolve' the background it overlays.

`DarkenCompositeOp`

Replace target image pixels with darker pixels from the foreground image.

LightenCompositeOp

Replace target image pixels with lighter pixels from the foreground image.

HueCompositeOp

Each pixel in the result image is the combination of the hue of the target image and the saturation and brightness of the foreground image.

SaturateCompositeOp

Each pixel in the result image is the combination of the saturation of the target image and the hue and brightness of the foreground image.

ColorizeCompositeOp

Each pixel in the result image is the combination of the brightness of the target image and the saturation and hue of the foreground image. This is the opposite of LuminizeCompositeOp.

LuminizeCompositeOp

Each pixel in the result image is the combination of the brightness of the foreground image and the saturation and hue of the target image. This is the opposite of ColorizeCompositeOp.

ScreenCompositeOp

Multiplies the inverse of each image's color information.

CopyCyanCompositeOp

Copy the cyan channel from the foreground image to the target image.

CopyMagentaCompositeOp

Copy the magenta channel from the foreground image to the target image.

CopyYellowCompositeOp

Copy the yellow channel from the foreground image to the target image.

CopyBlackCompositeOp

Copy the black channel from the foreground image to the target image.

ColorDodgeCompositeOp

Brightens the destination color to reflect the foreground color. Painting with black produces no change.

HardLightCompositeOp

Multiplies or screens the colors, dependent on the source color value. If the foreground color is lighter than 0.5, the destination is lightened as if it were screened. If the foreground color is darker than 0.5, the destination is darkened, as if it were multiplied. The degree of lightening or darkening is proportional to the difference between the foreground color and 0.5. If it is equal to 0.5 the destination is unchanged. Painting with pure black or white produces black or white.

SoftLightCompositeOp

Darkens or lightens the colors, dependent on the foreground color value. If the source color is lighter than 0.5, the destination is lightened. If the foreground color is darker than 0.5, the destination is darkened, as if it were burned in. The degree of darkening or lightening is proportional to the

difference between the foreground color and 0.5. If it is equal to 0.5, the destination is unchanged. Painting with pure black or white produces a distinctly darker or lighter area, but does not result in pure black or white.

The 'Paint' method

Argument: None

Result: None

Valid for: Any displayable object

The `Paint` method forces an object to be re-drawn immediately. You do not normally need to call this, since any changes made to an object automatically trigger a re-draw event.

The 'Paste' method

Argument: None

Result: None

Valid for: Edit, RichEdit, Document, Image

The `Paste` method pastes text from the clipboard into the Edit or Document object at the current insertion point, replacing the currently selected text (if any). This is what normally should happen when Paste is selected from the Edit menu. The `onChange` callback (if any) is triggered.

For a RichEdit control, the pasted text will include formatting information such as the font and text color.

For an Image object, the `Paste` method copies a picture from the clipboard into the Image object.

The 'Play' method

Argument: None

Result: Integer scalar - see text

Valid for: Movie

The `Play` method starts the current movie from the current position. If the movie is already playing, it has no effect. If the movie has ended, the current position will be at the end of the movie, so the `Play` method will have no visible effect unless the `Rewind` method is first called.

You typically use the `Play` method for a `Movie` object which does not have a controller (i.e. the `style` property is 1 or 2), in three main circumstances:

- For a movie which has been newly opened, to begin the movie;
- For a movie which has been paused using the `Stop` method, to resume playing from the current position;
- For a movie which has ended and which you want to repeat, you invoke the `Rewind` method and then the `Play` method.

The `Play` method returns a result which is a code as follows:

```
-1  Movie file not yet specified (or user chose Cancel in the movie
    selection dialog, or an error occurred in opening the file)
 1  Movie playing
 2  Movie not playing because it has ended
```

The 'Pointtocell' method

Argument: 2-element Numeric Vector

Result: 2-element Numeric Vector

Valid for: Grid

This method converts from coordinates of the grid control on the screen to a cell position. It is particularly useful for responding to mouse-down events on the grid, if you need to know which cell the user has clicked in.

The argument is a two-element vector of the vertical and horizontal position within the visible control (remembering that top, left is 0, 0). It is expressed in the current `scale` of the control; you will probably want to set the `scale` property to 5 for pixels before calling this method.

The result is an integer pair of Cell Row, Cell Column. For each element of pair, it returns -N if the point is in a header row or column, 0 if not in any row or column, +N if in normal row or column

The 'Pointtochart' method

Argument: 2-element Numeric Vector

Result: 2-element Numeric Vector

Valid for: Chart

This method converts from coordinates of the visible control on the screen to Chart coordinates (using the main Y scale). It is particularly useful for responding to mouse-down events on the chart, for example if you want to allow the user to select a point on the graph.

The argument is a two-element vector of the vertical and horizontal position within the visible control (remembering that top, left is 0, 0). It is expressed in the current `scale` of the control; you will probably want to set the `scale` property to 5 for pixels before calling this method.

The result is a two-element vector of the corresponding Y value (using the main Y scale) and X value of the Chart.

The 'Pointtochartalt' method

Argument: 2-element Numeric Vector

Result: 2-element Numeric Vector

Valid for: Chart

This method converts from coordinates of the visible control on the screen to Chart coordinates (using the alternate Y scale). It is particularly useful for responding to mouse-down events on the chart, for example if you want to allow the user to select a point on the graph, where the Series containing the point uses the Alternate Y scale.

The argument is a two-element vector of the vertical and horizontal position within the visible control (remembering that top, left is 0, 0). It is expressed in the current `scale` of the control; you will probably want to set the `scale` property to 5 for pixels before calling this method.

The result is a two-element vector of the corresponding Y value (using the alternate Y scale) and X value of the Chart.

The 'Popup' method

Argument: Either None, or a 2-element numeric vector

Result: None

Valid for: Menu

The `Popup` method is used for pop-up menus (i.e. menus which are not children of a window, and which thus do not appear in the window's menu bar). The `Popup` method displays the pop-up menu, and is typically called in response to a user action such as a right-mouse click.

If `Popup` is called without an argument, the menu pops up at the current mouse position. It can alternatively be called with a two-element numeric vector argument, specifying the Y and X position at which the menu should pop up on the screen. These values are expressed in the current `scale` units of the System object.

The 'Post' method

Argument: Character Vector

Result: 3-element Nested Vector

Valid for: HTTPClient

The `Post` method is similar to `Get`, and has the same syntax, arguments, and result structure. `Post` is used for HTML forms and pages where you need to send data to the web server; the argument has the data appended to the URL separated by a question mark, with `&` substituted for space (for example, `'http://www.blah.com?request=doit&code=101'`).

See the documentation on `Get` for more information.

The 'Poster' method

Argument: None

Result: None

Valid for: Movie

Under MacOS, movies can have a 'poster' defined for them, i.e. a still picture which represents the movie. The `Poster` method causes this picture to be displayed. If the movie is running, it will be stopped first.

Under Windows and Linux, this method simply displays the first frame of the movie.

This method should not be used for a Movie with controller (`style 0`).

The 'Preview' method

Argument: None

Result: None

Valid for: Movie

Under MacOS, movies can have a preview sequence defined, i.e. short extracts from the movie. The `Preview` method causes this to be played.

Under Windows and Linux, `Preview` simply plays the start of the movie.

This method should not be used for a Movie with controller (`style 0`).

The 'Print' method

Argument: See below for RichEdit

Result: None

Valid for: Printer, RichEdit, Browser, Chart

Printer object

Once you have started a print job using the `Open` method, you can cause text to be output to the printer by calling the `Print` method. The argument can be either a character vector (possibly with embedded carriage returns), or a text matrix. If you supply a matrix, a new line will be thrown at the end of each line of the matrix.

Printing takes place starting at the current value of the `position` property (relative to the top, left margin of the page). The `position` property is automatically updated after the method is called. If the bottom of the page is reached, a new page will automatically be started and printing will resume at the top of the next page. Note that text will *not* wrap round at the right margin.

The appearance of the text will depend on the current values of the `font` and `color` properties of the Printer object.

See also the `Draw` method which provides an alternative way of printing text as well as shapes and pictures.

RichEdit, Chart and Browser objects

For these controls, the `Print` method provides a quick way of printing the contents of the control. Usually it takes no argument, but (in the case of RichEdit) you can optionally provide a character vector which is the name of the print job.

The 'Receive' method

Argument: None

Result: Character vector

Valid for: Socket

The `Receive` method is used by client sockets to receive data from the server. It takes no arguments, and returns a character vector, which is the data received (as an untranslated sequence of raw bytes).

The APL task will block until the data has been received, or the time specified in the `timeout` property has elapsed.

If you do not want the task to be blocked waiting for data, use the `onReceive` event to ensure that `Receive` is called only when there is data already waiting.

If an error occurs, or the timeout expires, an empty vector will be returned. The `status` property can be used to find out more about the error,

The 'Refresh' method

Argument: None

Result: None

Valid for: Picture, Browser, OLEContainer

For a Browser, the `Refresh` method causes the current page to be re-loaded from the internet or from file.

For a Picture which is being used to display an Image (as a child object), the `Refresh` method causes the picture shown on screen to be updated from the off-screen bitmap of the Image object. Usually, this is automatic, unless you have changed the underlying ImageMagick object using `INA`.

For an OLEContainer control (Windows only), the `Refresh` method is used when the control is being used to display a linked document. It causes the display to be updated to reflect any changes to the source document.

The 'Reset' method

Argument: None

Result: None

Valid for: System object, GetMail

For the System object, the `Reset` method deletes all objects and resets the System Class sub-system to its initial state. The event queue is cleared.

For the GetMail object, the connection to the POP3 server is reset. Any pending deletes of e-mails on the server are aborted.

The 'Resize' method

Argument: None

Result: None

Valid for: Any object

The `Resize` method invokes the `onResize` callback (if any) for an object. It does not actually resize the object. Its main use is to trigger recalculation of dimensional information in your program, for example if the number of items displayed has changed and you want to ensure they are still evenly spaced.

The 'Rewind' method

Argument: None

Result: None

Valid for: Movie without controller

Where a `Movie` object does not have a controller (i.e. the `style` property is 1 or 2), the `Rewind` method resets the current position to the start of the movie. The movie can be restarted using the `Play` method.

The 'Save' method

Argument: Character vector, or a two-element nested vector

Result: None

Valid for: Chart, Image, RichEdit

Image

The `Save` method for an `Image` takes a character vector argument, which is the file name under which you want to save the image. You can specify the format in which you want to save it by setting the `format` property before calling this method, or alternatively allow the format to be determined automatically from the file extension. If you specify an empty vector for the filename, APLX displays a dialog allowing the user to specify the file name and type.

Chart

The `Save` method for a `Chart` causes the chart to be saved to a graphics file. The argument is the full path of the file to be written. Alternatively, it can be an empty vector, in which case a standard file dialog is displayed to allow the user to choose the name. The optional second parameter can be 0, meaning write the file using a platform-specific vector graphics format, or 1, meaning write the file in a bitmap format, or 2, meaning write the file in Scalable Vector Graphics (SVG) format. The default is 0.

If you want to save the chart in a format not supported by the `Chart` object, for example as a GIF, you can do this using an `Image` object. To do this, you need to read the `bitmap` property of the `Chart`, write this to the `bitmap` property of the `Image`, set the `Image` `format`, and then call the `Save` method of the `Image`.

RichEdit

The `Save` method causes APLX to save the contents of a `RichEdit` control to a file. The argument can be either a character vector containing the pathname of the file you wish to save to, or a nested vector where the first element is the file name and the second is a file type code. A type code of 0 (or omitting the type code) indicates that the contents should be saved as a plain text file. Under Windows, a type code of 1 can be used, to indicate that the contents should be saved as a Rich Text (.rtf) file, with embedded formatting information. Under MacOS, a type code of 2 means save as Styled text.

If you specify an empty vector as the file name, a dialog is displayed to allow the user to specify the file name and type.

The 'Screentoclient' method

Argument: 2-element integer vector

Result: 2-element integer vector

Valid for: Any visible control

This method converts coordinates relative to the whole screen to coordinates relative to the client area of the control. Both the argument and the result are two-element vectors of Row position, Column position (i.e. Y, X), always expressed in pixels. The `scale` properties of the control and `System` objects are ignored.

This method is particularly useful for relating mouse position coordinates reported in `⌈EV` to coordinates within the control.

The 'Send' method

Argument: None (Character Vector for Socket Object)

Result: None (Integer Scalar for Socket object)

Valid for: Any object, special meaning for Socket.

The `Send` method invokes the `onSend` callback (if any) for an object. The `onSend` callback is never invoked by APLX except in response to this method, so it can be used in any way which makes sense for your APL application.

For a `Socket`, the `Send` method has a special meaning. It takes a character vector argument, which is transmitted to the other end of the network connection. The data is sent exactly as it, without any character translation (i.e. the character vector is treated as a sequence of raw bytes). The return code is 0 for success, or 3 if an error occurs.

The 'SendMessage' method

Argument: None

Result: Integer Scalar

Valid for: `SendMail`

Once you have set up the properties of an e-mail using properties of the `SendMail` object, you send it by calling the `SendMessage` method (you must be connected to the SMTP server to do this). This method takes no argument; it returns an error code as follows: 0 = No error, 1 = Host not found, 2 = Authentication error, 3 = General or comms error, 4 = Attachment not found, 4 = Incomplete header (e.g. 'to' property not set), 5 = Recipient not found.

The 'Set' method

Argument: Nested vector of property-value pairs

Result: None

Valid for: Any object

The `set` method allows you to set multiple property values in a single statement, using `□WI` syntax (it cannot be used with dot notation). The argument is a nested vector of property names and values.

For example, the following sequence sets three properties for an 'OK' button:

```
'Win.But' □WI 'where' 5 9
'Win.But' □WI 'title' 'OK'
'Win.But' □WI 'enabled' 0
```

You can achieve the same result in one statement using the `set` method as follows:

```
'Win.But' □WI 'Set' ('where' 5 9) ('title' 'OK') ('enabled' 0)
```

The 'Setimages' method

Not currently implemented

The 'Setinfo' method

Argument: Nested vector

Result: None

Valid for: Tree

The `setinfo` method allows you to set the display attributes of one or more nodes in a Tree object. The first element of the argument is a character vector giving the attribute name, and should be one of 'label' 'list' 'images' 'forceparent' 'highlightbold' 'highlightcut' 'highlightdrop' 'highlightselect' 'expanded' or 'expandedonce'. (*Note:* The 'highlightcut', 'highlightdrop' and 'forceparent' keywords apply to Windows only, and are ignored under MacOS.)

The remaining element(s) give node identifiers and attribute values for those nodes. There are two possible ways to specify these:

(a) Element 2 is a vector of node identifiers, and element 3 is a vector/matrix of new values:

```
Mywin.Tree1.SetInfo 'highlightbold' (1 5 10) (1 0 1)
```

(b) Element 2 is a matrix where the first column contains the node identifiers and the remaining column(s) are the new values:

```
Mywin.Tree1.SetInfo 'highlightbold' (3 2 ρ 1 1 5 0 10 1)
```

The 'Setopacity' method

Argument: Scalar number between 0.0 and 1.0

Result: None

Valid for: Image

The `Setopacity` method allows you to specify that an image is semi-transparent. Making an image semi-transparent means that, when you place it on top of another image using the `Overlay` method, the background image will still be partially visible through the foreground image. The method takes a single argument, which is a number between 0.0 (completely transparent) to 1.0 (fully opaque, i.e. the background is not visible at all).

Note that some file formats, such as PNG, allow specific parts of the image to be transparent. In this case you don't need to set the opacity directly.

The 'Setpos' method

Not currently implemented

The 'Setup' method

Argument: None or Boolean Scalar

Result: Boolean scalar

Valid for: Printer

The `Setup` method displays the standard page-setup dialog, which allows the user to specify document-specific print parameters such as the paper size, paper source, and orientation. (This is the dialog which is normally displayed when the user selects Page Setup.. from the File menu of a typical application). It updates the current print record of the printer object, and should be called before calling the `Open` method which starts the print job.

Under Windows, there are two versions of this dialog. The default (invoked if you specify no argument, or an argument of 0) is simpler, and does not allow the user to set the page margins. The second (invoked if you specify an argument of 1) does allow the margins to be set by the user. Under MacOS or Linux, the same dialog is shown in both cases.

The return value indicates how the user ended the dialog. If the user pressed the OK button, it returns 1. A return value of 0 indicates that the Cancel button was pressed.

See also the `Job` method which displays the standard dialog for selecting print-job parameters such as the number of copies.

The 'Show' method

Argument: None

Result: None, or Integer scalar for pre-defined dialogs

Valid for: Any displayable object

For standard visible objects other than a Window, Form, Dialog, or Document, the `Show` method is equivalent to setting the `visible` property of an object to 1, i.e. it makes it visible if it was previously hidden. For a window-type object, it also makes the window come to the front if it was previously obscured by other windows.

For the pre-defined dialogs (`ChooseColor`, `ChooseFont`, `ChooseDir`, `OpenFile`, `SaveFile`, and `MsgBox`), the `Show` method displays the dialog in the modal state. The dialog remains open until the user selects a button which closes the dialog. The result of the `Show` method indicates which button was pressed.

For the ChooseColor, ChooseFont, ChooseDir, OpenFile, and SaveFile dialogs, the `Show` method returns 1 to indicate the OK button was pressed, or 0 to indicate the Cancel button. If the return value is 1, you should read the object's properties (such as the `color` property of the ChooseColor dialog) to find out what the user selected.

The possible return values for the MsgBox dialog depend on the `style` property which determines which of the buttons are displayed. The result will be one of:

```
1 = OK
2 = Cancel
3 = Abort
4 = Retry
5 = Ignore
6 = Yes
7 = No
```

The 'Showaboutbox' method

Argument: None

Result: None

Valid for: OCX controls (*Implemented under Windows only*)

The `Showaboutbox` method causes an OCX control to display its About... dialog (if any), typically giving copyright and version information about the control.

The 'Shownode' method

Argument: 2-element nested vector

Result: None

Valid for: Tree

The `Shownode` method scrolls the tree so that a particular node is displayed at the top, or (under Windows) makes a particular node the currently-selected one or the drag-and-drop target.

The argument is a two-element nested vector. The first element is a character vector, one of 'select', 'drop' or 'firstvisible'. The second element is the node identifier (or 0 or 'none' to remove the attribute from the currently selected node or drag-drop target).

The 'Showproperties' method

Argument: None

Result: Boolean scalar

Valid for: OCX controls and OLEContainer (*Implemented under Windows only*)

The `Showproperties` method causes an OCX control or the application associated with a document in an OLEContainer to display a dialog which allows the user to set properties associated with it. For example, if the OCX control displays a chart, the properties dialog would typically allow the user to select the type of chart, the scale, colors etc. (These properties would probably also be exported by the control, and thus could be changed directly by your APLX program).

The result returned is 1 if the user pressed OK, and 0 if the user pressed Cancel. However, due to a limitation in the Windows automation interface, a 1 will sometimes be returned even if the user pressed Cancel.

The 'Signal' method

Argument: Any

Result: None

Valid for: System and APL (child task) object

The `Signal` method allows the child task to send a message to the parent, and vice versa. It is used in conjunction with the `onSignal` event, as follows:

- If the parent wants to send a signal to the child, it invokes the `Signal` method in the APL (child task) object. This causes an `onSignal` event to trigger in the System object of the child task.
- Conversely, if the child task wishes to send a signal to the parent, it invokes the `Signal` method in its own System object. This causes an `onSignal` event to trigger in the APL (child task) object of the parent.

In both cases, the `Signal` method optionally takes an argument, which can be any APL array (or an APLX overlay created using `⌈0V`). This is typically used to send a command to the other task, or to return a result to it. Any argument to the `Signal` method is available to the receiving task as the `⌈WARG` system variable during the execution of the callback.

See the section on APLX Multi-tasking for more details.

The 'Sortchildren' method

Implemented under Windows only

Argument: See text

Result: None

Valid for: Tree

The `Sortchildren` method causes the children of a particular node to be sorted.

The argument is a two-element nested vector, or a character vector or integer scalar. The first element identifies the node, and can be a node ID or the character vector `'root'`. The second element is optional. If supplied, it should be `'recurse'` to indicate that the operation should be applied recursively to the children of the node.

The 'Stepit' method

Argument: None, or integer scalar

Result: None

Valid for: Progress

The `Stepit` method increments the current value of a Progress bar, increasing the length of the bar showing progress of the operation. If no argument is supplied, the value is incremented by the `increment` property. If an integer scalar argument is supplied, it is used as the increment.

The 'Stop' method

Argument: None

Result: None

Valid for: Movie, Browser

Where a Movie object does not have a controller (i.e. the `style` property is 1 or 2), the `Stop` method can be used to cause the playing of the movie to stop. If the movie is already stopped, it has no effect. The movie can be restarted from the same position by using the `Play` method.

For a Browser object, the `Stop` method aborts any pending page load which has not yet completed.

The 'Transform' method

Argument: See text

Result: None

Valid for: Image

The ImageMagick package which is accessed from the APLX Image object includes a very wide range of functions for transforming an image. For example, you can shear, scale, crop, rotate or skew the image, sharpen edges, adjust colors and intensity, and filter out noise. You can also apply special effects to the image, such as making it look like a charcoal drawing or an oil painting. The `Transform` method of the APLX Image object allows you to use many of these functions directly.

The syntax of the `Transform` method is:

```
ControlName.Transform 'Keyword' Arg1 Arg2...  
or  
'ControlName' □WI 'Transform' 'Keyword' Arg1 Arg2...
```

where *Keyword* selects the transformation you want to carry out, and the remaining arguments depend on the transformation (they are all numeric, with the scale always being pixels). Keywords are not case-sensitive. The following is a list of the supported transformations and arguments, together with the name of the underlying ImageMagick operation. See the ImageMagick documentation for full details of the operations and parameters.

Geometric transformations and scaling

Flip image about horizontal axis: (MagickFlipImage)

```
Object.Transform 'Flip'
```

Flip image about vertical axis: (MagickFlopImage)

```
Object.Transform 'Flop'
```

Scale image to twice current size: (MagickMagnifyImage)

```
Object.Transform 'Magnify'
```

Scale image to half current size: (MagickMinifyImage)

```
Object.Transform 'Minify'
```

Scale image to specified X Y size: (MagickScaleImage)

```
Object.Transform 'Scale' x y
```

Roll image: (MagickRollImage)

```
Object.Transform 'Roll' x_offset y_offset
```

Extract a rectangular region from the image: (MagickCropImage)

```
Object.Transform 'Crop' width height x y
```

Rotate image clockwise by specified number of degrees: (MagickRotateImage)

```
Object.Transform 'Rotate' degrees
```

Shear the image along X or Y axis, or both: (MagickShearImage)

```
Object.Transform 'Shear' x_shear y_shear
```

Image enhancement and adjustment

Reduce the speckle noise in an image: (MagickDespeckleImage)

```
Object.Transform 'Despeckle'
```

Apply digital filter to improve noisy image: (MagickEnhanceImage)

```
Object.Transform 'Enhance'
```

Apply Median filter to improve noisy image: (MagickMedianFilterImage)

```
Object.Transform 'MedianFilter' radius
```

Equalize the image color histogram: (MagickEqualizeImage)

```
Object.Transform 'Equalize'
```

Enhance the contrast of a color image: (MagickNormalizeImage)

```
Object.Transform 'Normalize'
```

Sharpen an image: (MagickSharpenImage)

```
Object.Transform 'Sharpen' radius sigma
```

Sharpen image with a Gaussian operator: (MagickUnsharpMaskImage)

```
Object.Transform 'UnsharpMask' radius sigma amount threshold
```

Enhance edges using a convolution filter of the given radius: (MagickEdgeImage)

```
Object.Transform 'Edge' radius
```

Smooth contours of image, preserving edge information: (MagickReduceNoiseImage)

```
Object.Transform 'ReduceNoise' radius
```

Adjust brightness, saturation, and hue of the image: (MagickModulateImage)

```
Object.Transform 'Modulate' brightness saturation hue
```

Gamma-correct an image: (MagickGammaImage)

```
Object.Transform 'Gamma' gamma
```

Special effects

Implode pixels by specified amount: (MagickImplodeImage)

```
Object.Transform 'Implode' amount
```

Make image look like an oil painting: (MagickOilPaintImage)

```
Object.Transform 'OilPaint' radius
```

Make image look like a charcoal drawing: (MagickCharcoalImage)

```
Object.Transform 'Charcoal' radius sigma
```

Add three-dimensional effect to grayscale image: (MagickEmbossImage)

```
Object.Transform 'Emboss' radius sigma
```

Randomly displace each pixel in a block: (MagickSpreadImage)

```
Object.Transform 'Spread' radius
```

Swirl the pixels about the center of the image: (MagickSwirlImage)

```
Object.Transform 'Swirl' degrees
```

Change pixel values depending on intensity relative to threshold: (MagickThresholdImage)

```
Object.Transform 'Threshold' threshold
```

Apply effect similar to exposing photo paper to light: (MagickSolarizeImage)

```
Object.Transform 'Solarize' threshold
```

Blur image: (MagickBlurImage)

```
Object.Transform 'Blur' radius sigma
```

Blur image (Gaussian blur): (MagickGaussianBlurImage)

```
Object.Transform 'GaussianBlur' radius sigma
```

Blur image radially: (MagickRadialBlurImage)

```
Object.Transform 'RadialBlur' angle
```

Simulate motion blur: (MagickMotionBlurImage)

```
Object.Transform 'MotionBlur' radius sigma angle
```

Create ripple effect in image: (MagickWaveImage)

```
Object.Transform 'Wave' amplitude wave_length
```

Adjust levels of image: (MagickLevelImage)

```
Object.Transform 'Level' blackpoint gamma whitepoint
```

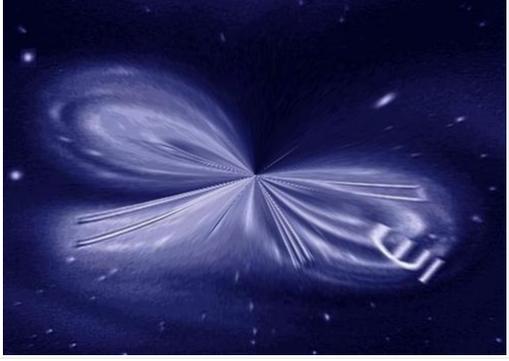
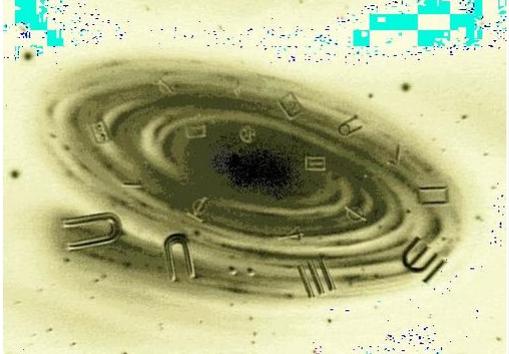
Force all pixels below threshold to black: (MagickBlackThresholdImage)

```
Object.Transform 'BlackThreshold' threshold
```

Force all pixels above threshold to white: (MagickWhiteThresholdImage)

```
Object.Transform 'WhiteThreshold' threshold
```

Examples of image transformations

<p><i>Original image</i></p>	
<p>Obj.Transform 'Flop'</p>	
<p>Obj.Transform 'Implode' 1</p>	
<p>Obj.Transform 'Solarize' 0.3</p>	

The workspace 10 HELPTRANSFORM has a demonstration of these and other image transformations.

The 'Trigger' method

Argument: Character vector containing a callback property name

Result: None

Valid for: Any control

The `Trigger` method provides a general way for invoking a callback function, and thus in a sense can be used to simulate any event. It allows your program to cause a callback associated with a particular object and event to run. It takes as an argument the name of an event, and triggers the callback (if any) for that event. For example, to invoke the Double Click handler callback for a List object, you could enter:

```
MyWin.List.Trigger 'onDb1Click'
```

Note that running this method invokes the callback function, but does not otherwise affect the object concerned. Using the `Trigger` method can help avoid duplication of code. It also allows you to create custom controls by taking over event handling, and triggering higher-level events where appropriate (see the next chapter for further information on this subject). For example, you might want to achieve the effect of a custom Check box control which changed color when selected. To do this, you might define an `onMouseDown` callback function, which handled the color change, and which then called the `Trigger` method to create an `onClick` event for the object.

The 'Undo' method

Argument: None

Result: None

Valid for: Edit, RichEdit, Document

The `Undo` method undoes the last change made by the user. If there is no change which can be undone, it does nothing. (See also the `canundo` property).

The 'Valueof' method

Argument: Character vector

Result: Usually integer, but can be any type

Valid for: OCX controls, OLE server applications and OLEContainer (*Implemented under Windows only*)

Many OCX controls and OLE servers define named constant values or 'enums' as valid arguments for properties and methods. Often the documentation will refer to the names of these constants without giving the values, which you need to use them from APL. (The APLX Control Browser window shows constant values by category for controls and servers which publish them.)

The `Valueof` method allows you to determine the value of a constant associated with the control or server at runtime. It takes a character vector argument which is the name, and returns the value. For example, Formula One defines a set of enumerated values for file formats. To select the Formula One Version 3 format, you need the constant `F1FileFormulaOne3`. You can determine it from within APLX as follows:

```
W.F1.ValueOf 'F1FileFormulaOne3'  
5
```

The 'Wait' method

Argument: None

Result: None

Valid for: Window, Form or Dialog

The `Wait` method is used to make a Window or Dialog modal, i.e. to disable all other application windows until the user has responded to the dialog. The modal state remains in force until the window is closed, hidden or destroyed. If the window is closed or hidden when the `Wait` method is called, it will first be opened and/or made visible. Note that your program still continues execution, and continues to handle events (via `QWE`) as usual. If you want your program to wait for the modal dialog to complete, then immediately after invoking the `Wait` method you should call `QWE` with a character right argument which is the name of the window.

The session window and other APL debugging windows are not disabled by this method, so you can interrupt the APL program as usual.

Section 5. System Classes: Callbacks

The 'onAboutMenu' callback

Event generated under MacOS only

Event number: 37
Generated for: System

When you create a packaged application from your APLX workspace, your code will normally use System Classes to create windows and the menus associated with them. However, under MacOS, the 'About..' menu item resides in the System menu (or application menu under MacOS X), so it is outside the scope of the user-defined menus.

The `onAboutMenu` callback allows your packaged APL application to be notified when the user has selected the 'About..' menu item, so that you can display an appropriate information window. This will typically contain information about the application, any copyright notice, and a version number.

Under Windows, this callback will never be triggered. Instead, you should create your own 'About..' menu item, usually in your own 'Help' menu.

See also the `onPreferencesMenu` and `onQuitMenu` callbacks.

The 'onActivate' callback

This is a synonym for `onFocus`.

The 'onChange' callback

Event number: 23
Generated for: Combo, Edit, RichEdit, Document, Scroll, Selector, Splitter, Spinner, Trackbar, Grid

The `onChange` callback is generated when the text of an Edit, Combo, RichEdit or Document object has changed. This may be either because the user has typed something, or because your application has changed the text under program control. Generally, in your `onChange` callback handler you will read the text property of the object, and take action according to the new value (for example, disable the OK button if the text is not currently valid). Note that the `onChange` callback will be triggered every time the user types when the object concerned has focus. If you wish to validate the user's input only when it is complete, it is better to do this by means of an `onUnfocus` callback.

For a Scroll, Spinner or Trackbar control, the `onChange` callback is generated when the user changes the control's value. For a Splitter, it is invoked when the user moves the Splitter so that the attached controls are re-sized. For a Selector, it is invoked when the user selects a different Page.

For a Grid object, the `onChange` callback is triggered when the user has finished editing a cell. You can use it to validate the new cell contents, or to recalculate other cells in the Grid. The event-specific parameters in `EV` are (in index origin 1):

```
EV[6]    Row number of the edited cell
EV[7]    Column number of the edited cell
EV[8]    Flag to indicate whether the contents were valid
```

The 'onClick' callback

Event number: 24

Generated for: All visible controls

The `onClick` callback is run after the user has pressed the mouse button in the object (a `MouseDown` event will be created first). It indicates that the object has been selected (or de-selected, in the case of a Radio button). It is also triggered for a List if the user changes the selected item using the keyboard, and for an Edit, RichEdit or Document object if the selection/insertion point changes as a result of user input.

The 'onClose' callback

Event number: 10

Generated for: All objects, but usually applies to Window, Form, Document or Dialog, or an APL (child task) object.

The `onClose` callback indicates a request to close an object (usually a window). It can be generated as a result of the user clicking the close box of a window, or by the system. If you have defined an `onClose` callback, the window is not closed automatically; your `onClose` callback should do this if appropriate, by invoking the `Close` method with an argument of 1 to cause an unconditional close. You might also wish to delete the window as well.

The `onClose` callback is treated specially in one way. Normally when an APL callback is defined for an event, it is run after the system default handler for that event. The default handler for a Close event is to close and delete the object, but doing this would make it impossible for your `onClose` handler to suppress the close action. For this reason, setting the `onClose` callback property automatically disables the default system handler for the close event on that object.

For an APL (child task) object, the `onClose` callback is triggered when the child task is about to terminate for any reason.

The 'onColMoved' callback

Event number: 45
Generated for: Grid

This event is triggered when the user has moved a column by dragging it to a different position in the grid. (This is possible only if the `style` property includes the flag 1024, column moving allowed). The event-specific parameters in `⊞EV` are (in index origin 1):

```
⊞EV[6]    Column number before the move
⊞EV[7]    The position of the column after the move
```

The 'onConnectRequest' callback

Event number: 49
Generated for: Socket

This event occurs on a server-side socket, when you have called `Listen` and a client tries to connect. It indicates that you should now call the `Accept` method to accept the connection.

The 'onDbClick' callback

Event number: 25
Generated for: All visible controls

If the user double-clicks in a control, the `onDbClick` callback will be run (a `Click` event is generated first).

For example, your APL program might interpret a double-click on a List box in a dialog as being as equivalent to pressing the OK button.

The 'onDeactivate' callback

This is a synonym for `onUnfocus`

The 'onDestroy' callback

Event number: 27

Generated for: Any object except the System object

When an object is deleted, the `onDestroy` callback (if any) for it is triggered. However, the object itself no longer exists when the callback function runs, and so it is too late to reference any properties of the object. For the same reason, `THIS` cannot be used in an `onDestroy` callback.

The 'onDisconnect' callback

Event number: 52

Generated for: Socket

This event occurs both on clients and servers. It indicates that the other end has closed the connection. You should now call `close`, and delete the socket.

The 'onDragDrop' callback

Event number: 34

Valid for: Any visible control

The `onDragDrop` callback is invoked for the target control when a successful drag-and-drop operation has been completed, and the user releases the mouse over the target control. (For this to happen, the `sourceformats` property of the source must be compatible with the `targetformats` property of the target).

Under Windows, it will typically be preceded by an `onDragEnter` and several `onDragOver` events. The source control will be notified by an `onDragEnd` event.

The `dragsource` property will contain the name of the control being dragged.

The event-specific parameters in `⌈EV` are (in index origin 1):

`⌈EV[6]` The 'tie' property value of the source control being dragged

The 'onDragEnd' callback

Event number: 32

Valid for: Any visible control

The `onDragEnd` callback occurs for the *source* control when a drag-and-drop operation has been completed (the target control will receive an `onDragDrop` event).

The name of the target control (on to which the source control is being dragged) will be in the `droptarget` property.

The event-specific parameters in `⌈EV` are (in index origin 1):

`⌈EV[6]` The 'tie' property value of the target control to which the source control was dragged

The 'onDragEnter' callback

Event generated under Windows only

Event number: 35

Valid for: Any visible control

The `onDragEnter` callback is invoked for the target control when a potential drag-and-drop operation begins and the mouse enters the target control. (A potential drag-and-drop operation occurs when the user drags a *source* control onto a *target* control and the `sourceformats` property of the source is compatible with the `targetformats` property of the target, but before the user releases the mouse to complete the operation).

The callback will typically be followed by `onDragOver` events as the user moves the mouse, and possibly an `onDragLeave` event if the mouse leaves the control. It may be followed by an `onDragDrop` event if the user releases the mouse button over the target control.

The `dragsource` property will contain the name of the control being dragged.

The event-specific parameters in `⊞EV` are (in index origin 1):

`⊞EV[6]` The 'tie' property value of the source control being dragged

The 'onDragLeave' callback

Event generated under Windows only

Event number: 36

Valid for: Any visible control

The `onDragLeave` callback is invoked for the target control when a potential drag-and-drop operation has begun, but the user moves the mouse out of the target control. (A potential drag-and-drop operation occurs when the user drags a *source* control onto a *target* control and the `sourceformats` property of the source is compatible with the `targetformats` property of the target, but before the user releases the mouse to complete the operation).

The `dragsource` property will contain the name of the control being dragged.

The event-specific parameters in `⊞EV` are (in index origin 1):

`⊞EV[6]` The 'tie' property value of the source control being dragged

The 'onDragOver' callback

Event generated under Windows only

Event number: 33

Valid for: Any visible control

The `onDragOver` callback is invoked for the target control when a potential drag-and-drop operation has begun, and the user moves the mouse over the target control. (A potential drag-and-drop operation occurs when the user drags a *source* control onto a *target* control and the `sourceformats` property of the source is compatible with the `targetformats` property of the target, but before the user releases the mouse to complete the operation).

It will typically be preceded by an `onDragEnter` event, and possibly followed by an `onDragLeave` or `onDragDrop` event.

The `dragsource` property will contain the name of the control being dragged.

The event-specific parameters in `⌈EV` are (in index origin 1):

`⌈EV[6]` The 'tie' property value of the source control being dragged

The 'onDragStart' callback

Event generated under Windows only

Event number: 31

Valid for: Any visible control

The `onDragStart` callback is invoked for the source control when the user starts dragging the control. At this stage, there is no target for the drag-and-drop operation.

The 'onDraw' callback

This is a synonym for the `onPaint` callback property.

The 'onDropDown' callback

This is a synonym for the `onPopUp` callback property.

The 'onError' callback

Event number: 41

Generated for: Browser and APL (child task) object

Browser

For a Browser object, the `onError` callback is invoked if an error occurs when the Browser tries to load a page.

APL Object

The `onError` callback is invoked in an `APL` object belonging to a parent task when an untrapped error occurs during function execution in a child task. The error message (in the same format as `⌈ERM`) will be available in `⌈WARG` during the callback.

The event-specific parameters in `⌈EV` are (in index origin 1):

`⌈EV[6]` Task ID for the child task

See the section on APLX Multi-tasking for more details.

The 'onExecute' callback

Event number: 44

Generated for: `APL` (child task) object

The `onExecute` callback is invoked in an `APL` object belonging to a parent task when the child task is about to execute an expression or command (either typed by the user, or invoked under program control using the `Execute` method).

During the callback, `⌈WARG` contains the command or expression which is about to be executed, as a character vector. This can be used for example to write an `APL` tutorial application when the parent task can see what the user is entering in the child-task session window.

The event-specific parameters in `⌈EV` are (in index origin 1):

`⌈EV[6]` Task ID for the child task

See the section on APLX Multi-tasking for more details.

The 'onFocus' callback

Event number: 7

Generated for: Window, Dialog, Document, plus any control which can receive the input focus

Synonym: `OnActivate`

The `onFocus` callback is invoked when the object gets the input focus. For a Window, Form, Dialog or Document, this means when the window is activated (for example, by the user clicking in it when it is not the front window). In addition, when the window is activated, the currently-active control (if any)

on the window will also receive a Focus event. This will also happen within a window if the user tabs or clicks between controls.

The `onFocus` callback is invoked irrespective of whether the focus was gained through user action or under program control (for example, by calling the `Focus` method).

The 'onHide' callback

Event number: 29

Generated for: Any visible object

The `onHide` callback will be run if an object (or its parent) is hidden using the `Hide` method or by setting the `visible` property to 0.

The 'onKeyDown' callback

Event number: 1

Generated for: Any control which can receive the input focus

A Key Down event is generated when the user presses a key whilst the object has the input focus. The event-specific parameters in `⎕EV` are (in index origin 1):

<code>⎕EV[6]</code>	Raw ASCII code for this key
<code>⎕EV[7]</code>	APL's translated code for this key
<code>⎕EV[8]</code>	Scan code (ie key number)
<code>⎕EV[9]</code>	Modifiers: (0=unshifted, 1=shifted, 2=alt/option, 4=control, 8=command, and sums thereof)

The 'onKeyPress' callback

Event number: 30

Generated for: Any control which can receive the input focus

A Key Press event is generated when the user presses a key which produces a character, whilst the object has the input focus. (It is generated after the corresponding Key Down event). The event-specific parameters in `⎕EV` are (in index origin 1):

<code>⎕EV[6]</code>	ASCII code for this key and modifier keys
<code>⎕EV[7]</code>	APL's translated code for this key

The 'onKeyUp' callback

Event number: 2

Generated for: Any control which can receive the input focus

A Key Up event is generated when the user releases a key whilst the object has the input focus. The event-specific parameters in `⌈EV` are (in index origin 1):

<code>⌈EV[6]</code>	Raw ASCII code for this key
<code>⌈EV[7]</code>	APL's translated code for this key
<code>⌈EV[8]</code>	Scan code (ie key number)
<code>⌈EV[9]</code>	Modifiers: (0=unshifted, 1=shifted, 2=alt/option, 4=control, 8=command, and sums thereof)

The 'onLimit' callback

Event number: 22

Generated for: Edit

If the user attempts to enter more characters into an Edit field than you have specified using the `limit` property, the `onLimit` callback will be run.

The 'onMenu' callback

Event number: 6

Generated for: Window, Form, Dialog, Document

The currently-active window receives a Menu event if the user selects a menu from the main menu bar. This can be used as alternative way of detecting menu clicks, instead of using the menu item's `onClick` callback. The event-specific parameters in `⌈EV` are (in index origin 1):

<code>⌈EV[6]</code>	Menu number in the menu bar
<code>⌈EV[7]</code>	Item number within the menu

Menu and item numbers start at 1, and are in the sequence in which they appear (the same as the `order` property of the Menu object). Invisible menu items, and separators, are counted in the sequence. If you have a sub-menu, the item number is multiplied by 100 for each level down the hierarchy.

For example, if you have a File menu as the first menu in the menu bar, and it has two items Open and Save, the Open menu will be reported as Menu 1 Item 1, and the Save item as Menu 1 Item 2. If the Save menu has a sub-menu with items Final and Draft, these will be reported as Menu 1 Item 201 and Menu 1 Item 202 respectively.

The 'onMouseDown' callback

Event number: 3

Generated for: Any visible, enabled object (except as described below)

When the user presses the mouse button in the enclosing rectangle of an object, the onMouseDown callback (if any) is invoked (note that for standard controls such as Buttons and Lists, you should normally rely on the Click event rather than the Mouse Down event to detect user input). It can be used to make non-standard controls responsive to mouse input (for example, a Rectangle or Static text). The event-specific parameters in `⌘EV` are (in index origin 1):

<code>⌘EV[6]</code>	Mouse vertical position in window coordinates
<code>⌘EV[7]</code>	Mouse horizontal position in window coordinates
<code>⌘EV[8]</code>	Button number (always 1 on the Macintosh)
<code>⌘EV[9]</code>	Modifiers (as for Key Down)

Note: Under MacOS, this event is not generated for certain controls, including Button, Radio and Check controls.

The 'onMouseMove' callback

Event number: 15

Generated for: Any visible control

A Mouse Move event is generated when the user moves the mouse over a control. The event-specific parameters in `⌘EV` are (in index origin 1):

<code>⌘EV[6]</code>	Mouse vertical position in window pixel coordinates
<code>⌘EV[7]</code>	Mouse horizontal position in window pixel coordinates
<code>⌘EV[8]</code>	0
<code>⌘EV[9]</code>	Modifiers: (0=unshifted, 1=shifted, 2=alt/option, 4=control, 8=command, and sums thereof)

The 'onMouseUp' callback

Event number: 4

Generated for: Any visible, enabled object

When the user releases the mouse button in the enclosing rectangle of an object, the `onMouseUp` callback (if any) is invoked. The event-specific parameters in `⊞EV` are the same as for Mouse Down.

The 'onMove' callback

Event number: 11

Generated for: Form, Window, Document, Dialog

The `onMove` callback is invoked when the user moves a window. The event-specific parameters in `⊞EV` are (in index origin 1):

<code>⊞EV[6]</code>	New top position
<code>⊞EV[7]</code>	New left position

The 'onMovieEnd' callback

Event number: 21

Generated for: Movie without controller

When a Movie object is playing a movie and the end of the movie is reached, the `onMovieEnd` callback is invoked (unless the movie is in continuous play mode in which case the movie never ends). The `onMovieEnd` callback is not applicable for a movie with controller (`style 0`).

The 'onOpen' callback

Event number: 26

Generated for: Any object

The `onOpen` callback is invoked after an object has been opened.

For the System object, the `onOpen` event has a special meaning. If you have packaged your APL workspace into a standalone application, this event is triggered under MacOS when the user drags a file icon on to your application icon, once the application is running. (It is not triggered when the application first starts up, and is not triggered under Windows). The `file` and `action` properties indicate what files you need to open or print.

The 'onPaint' callback

Event number: 9
Generated for: Any visible object
Synonym: `onDraw`

Normally, APL takes care of all drawing and updating of an object on the screen. However, the `onPaint` callback can be used to supplement (or even to replace) the default drawing method for given object. It is invoked whenever the object needs to be redrawn for any reason (for example, if a property has been changed, or if the window was previously obscured and now has been uncovered).

The 'onPopup' callback

Event number: 19
Generated for: Menu

The `onPopup` callback is invoked when the menu is popped up.

The 'onPreferencesMenu' callback

Event generated under MacOS X only

Event number: 38
Generated for: System

When you create a packaged application from your APLX workspace, your code will normally use System Classes to create windows and the menus associated with them. However, under MacOS X, the 'Preferences' menu item resides in the application menu which MacOS X creates, so it is outside the scope of the user-defined menus.

The `onPreferencesMenu` callback allows your packaged APL application to be notified when the user has selected the 'Preferences' menu item, so that you can display an appropriate dialog.

Under Windows and MacOS 8 or 9, this callback will never be triggered. Instead, you should create your own 'Preferences' menu item.

See also the `onAboutMenu` and `onQuitMenu` callbacks.

The 'onQuitMenu' callback

Event generated under MacOS X only

Event number: 39
Generated for: System

When you create a packaged application from your APLX workspace, your code will normally use System Classes to create windows and the menus associated with them. However, under MacOS X, the 'Quit' menu item resides in the application menu which MacOS X creates, so it is outside the scope of the user-defined menus.

The `onQuitMenu` callback allows your packaged APL application to be notified when the user has selected the 'Quit' menu item, so that you can do any necessary cleaning up and exit the application.

Under Windows and MacOS 8 or 9, this callback will never be triggered. Instead, you should create your own 'Quit' menu item, usually in the 'File' menu.

See also the `onAboutMenu` and `onPreferencesMenu` callbacks.

The 'onReady' callback

Event number: 42
Generated for: Browser and APL (child task) object

Browser

For a Browser object, the `onReady` callback is invoked when a new page has been loaded.

APL object

The `onReady` callback is invoked in an APL object belonging to a parent task when the state of the child task changes from executing to ready for input.

The event-specific parameters in `⌵EV` are (in index origin 1):

`⌈EV[6]` Task ID for the child task

See the section on APLX Multi-tasking for more details.

The 'onReceive' callback

Event number: 50
Generated for: Socket

This event occurs both on clients and servers. It indicates that the other end has sent some data, which you can now read using the `Receive` method.

The 'onResize' callback

Event number: 12
Generated for: Any visible object (usually windows)

The `onResize` callback is called when the user resizes a window, or when your program calls the `Resize` method. The event-specific parameters in `⌈EV` apply to a Window, Form, Dialog, or Document only and are (in index origin 1):

`⌈EV[6]` New width in pixel units
`⌈EV[7]` New height in pixel units

Note that this callback is not invoked if your program changes an object's size by setting the `size` or `where` property - you need to call the `Resize` method as well if you want the callback function to be run.

In older APL systems, this callback was frequently used to re-arrange the controls in a window to reflect the re-sizing. In APLX, you can often do this much more simply by specifying the `align` or `anchors` properties of the controls in the window.

The 'onRowMoved' callback

Event number: 47
Generated for: Grid

This event is triggered when the user has moved a row by dragging it to a different position in the grid. (This is possible only if the `style` property includes the flag 512, row moving allowed). The event-specific parameters in `OEV` are (in index origin 1):

```
OEV[6]    Row number before the move
OEV[7]    The position of the row after the move
```

The 'onScroll' callback

Event number: 14
Generated for: Grid

The `onScroll` callback indicates that a Grid control has been scrolled. The visible position after the scroll can be determined using the `firstvisible` or `view` properties.

The 'onSelChange' callback

Event number: 40
Generated for: RichEdit, Grid, Tree

For a RichEdit control, this event is triggered when the user changes the selection point.

For a Tree control, it is triggered when the user changes the selected node.

For a Grid control, this event is triggered when the user selects a new cell. The event-specific parameters in `OEV` are (in index origin 1):

```
OEV[6]    Row number of the newly-selected cell
OEV[7]    Column number of the newly-selected cell
```

The 'onSelection' callback

Alternative name for the `onClick` callback

The 'onSend' callback

Event number: 20
Generated for: Any object

The `onSend` callback is used for inter-object or inter-task communication. It indicates that the `Send` method has been invoked for the object.

The 'onShow' callback

Event number: 28
Generated for: Any object

The `onShow` callback will be run if an object (or its parent) is made visible using the `Show` method or by setting the `visible` property to 1.

The 'onSignal' callback

Event number: 43
Generated for: System and APL (child task) objects

The `onSignal` callback allows Parent and Child tasks to send messages to each other, with associated data. It is invoked in an APL object belonging to a parent task when the child task executes the `Signal` method of its `System` object. Conversely, it is invoked in the `System` object of a child task when the Parent executes the `Signal` method of its APL child task object.

During the callback, `⍵WARG` contains the data (which can be any APL array or overlay) which the calling task passed as the argument to the `Signal` method.

The event-specific parameters in `□EV` are (in index origin 1):

`□EV[6]` Task ID for the task which sent the signal

See the section on APLX Multi-tasking for more details.

The 'onStop' callback

Alternative name for the `onMovieEnd` callback

The 'onTimer' callback

Event number: 18
Generated for: Timer

The `onTimer` callback is run at regular intervals by a Timer object (the frequency being set by the `interval` property).

The 'onUnfocus' callback

Event number: 8
Generated for: Window, Form, Dialog, Document, and any control which can have the input focus
Synonym: `onDeactivate`

The `onUnfocus` callback is called when an object loses the input focus (either because the user has done something, or under program control). It is typically used to validate user input in Edit controls.

The 'onZoom' callback

Not implemented under Windows and Linux

Event number: 13
Generated for: Window, Document, Dialog

When the user clicks in the Zoom box of a window, the default system action is to resize the window appropriately and create a Resize event. However, if you wish to override this behaviour you can define an `onZoom` callback which does something different (for example, zooms the window to a size which you determine dynamically). The event-specific parameters in `EV` are (in index origin 1):

<code>EV[6]</code>	Current window width in pixels
<code>EV[7]</code>	Current window height in pixels
<code>EV[8]</code>	Reason code (1 for normal size, 2 for full size)

Section 6. Using the Draw Method

Using the Draw method

The `Draw` method allows you to draw text, lines, patterns, pictures and geometric shapes on windows, controls, and printer pages. It takes an argument which comprises one or more phrases which each start with a keyword indicating the operation to perform, followed by the arguments to that operation. (see the workspace `10_HELPDRAW` for examples).

Drawing takes place using a current `Pen` (for foreground drawing) and `Brush` (for background drawing). You can also set the `Mode` which determines how drawing interacts with what is already on the screen.

Coordinates are interpreted according to the current *Draw-method Scale* for the object. The first time that the `Draw` method is used on a control, this is set to be the same as the `scale` property of the control. However, you can change it to be either one of the standard fixed scales, or to be proportional to the window size.

APLX automatically handles window refreshing for you if the window is uncovered or resized (you can switch this off using the `auto redraw` property). It does this by storing the sequence of `Draw` commands applicable to the window or control, and replaying them when an update is required. You can also group a series of commands together, and selectively enable or disable them (for example, to temporarily hide the labels on a graph), or delete them altogether from the saved sequence. This is useful for animation effects.

The general syntax of the `Draw` method is:

```
Control.Draw '<Keyword>' Arg1 Arg2...
```

where *Control* is usually the a reference to the Window into which you want to draw, or a `Printer`, or a `Frame`, `Picture` or `Image` object, identified using dot notation (for example, `Win.Pic.Image1`). You can also supply multiple sequences of commands on one line:

```
Control.Draw ('<Keyword1>' Arg1) ('<Keyword2>' Arg1 Arg2)...
```

Alternatively, you can use the equivalent `QWI` syntax:

```
ControlName QWI 'Draw' '<Keyword>' Arg1 Arg2...
ControlName QWI 'Draw' ('<Keyword1>' Arg1) ('<Keyword2>' Arg1 Arg2)...
```

where *ControlName* is a character vector containing the name of the window or control (including parent if applicable), for example: `'Win'` or `'Win.Pic.Image1'`

The keywords are not case-sensitive, but it is recommended that you enter them in the case shown in the following sections.

The operations you can carry out using the `Draw` method fall into four categories:

- Operations which set the state of the drawing sub-system for the current control, defining how subsequent drawing commands will work. These include commands to set the Pen, Brush, Font and Scale.
- Operations which cause drawing to take place on the current control. These include commands to draw lines, rectangles, arcs, and polygons, and to draw text (and find out what size rendered text would require on the screen). You can also draw bitmaps and other images, either from file or directly as an array of color values held in an APL variable.
- Operations which determine how drawing commands are replayed when a window update is required. These include commands to group graphic elements together, to enable and disable specific groups, and to delete commands from the saved list. You can also obtain a bitmap or Scalable Vector Graphics (SVG) representation of the drawing, or copy it to the Clipboard.

In the descriptions which follow, optional parameters are shown in square brackets.

Draw method: State commands

Default

Syntax: `Control.Draw 'Default'`

Sets the Font, Pen, Brush and Scale to the default values for further draw commands, but does not delete the existing saved sequence of commands. The default Pen is a black solid line of 1 pixel width. The default Brush is the window background color. The default Font is Arial 10 point, plain style. The default text foreground color is black and the default text writing mode is transparent. The draw-method scale is set to the scale of the parent object at the time when the `Draw` method was first used.

Scale

Syntax 1: `Control.Draw 'Scale' Value`

Syntax 2: `Control.Draw 'Scale' Height Width`

Syntax 3: `Control.Draw 'Scale' Top Left Height Width`

Syntax 4: `Control.Draw 'Scale' 0 Top Left Height Width`

Sets the Draw-method scale for further drawing commands.

When the `Draw` method is first used on a control or window, the Draw-method scale is set to be the same as the `scale` property of the control or window. The top, left of the drawing area is at position 0 0, and the coordinates increase in the downwards and rightwards directions. Subsequent changes to the `scale` property of the control or window have no effect on the `Draw` method, but you can use the `'Scale'` keyword in the `Draw` method to select a different scale for subsequent `Draw` method commands.

In the first syntax form shown above, the `Value` parameter selects one of the standard scales:

1. Use 'character' units. A character unit has the same height as a line of text in the default system font (16 points), and the width of a typical character (8 points). In this case the vertical and horizontal scales are different.
2. Use 'dialog' units, (1/8 system-font high, 1/4 system-font wide)
3. Use point units (nominally 72 points per inch)
4. Use 'twips' units (nominally 1440 twips per inch)
5. Use pixel units (also usually 72 per inch on the Macintosh), but this depends on the resolution and screen size.

6. Use nominal millimetres. Depending on the screen size, this may not be accurate for controls displayed on a screen, but it should be accurate when printing on paper (provided the printer driver is set up correctly).
7. (*Applicable to Chart objects only*) Use the same scale as the axes of the Chart. This is very useful for applying Draw method commands to a chart, for example to add lines showing confidence limits or annotate certain values with text. By using scale 7, you can ensure that the Draw commands always match the chart even if it is resized or re-scaled.

In the remaining syntax forms, the scale is proportional to the window or control size. The Height and Width parameters are used to indicate the number of units corresponding to the height and width of the window or control. For example, if Height is 200 and Width is 400, then the scale will be such that there are 200 vertical units between the top and bottom of the window, and 400 units across. This will remain true even if the window is resized, i.e. the scale is proportional to the window size, so that the graphic will be shrunk or stretched automatically when the window is re-drawn.

If the Top and Left parameters are supplied, the drawing origin is shifted such that these parameters correspond to the top left point of the drawing area.

Note that you can specify negative parameters for any of Height Width Top or Bottom. If you use a negative value for Height or Width, it changes the direction of the coordinate system. For example, the command

```
Win.Draw 'Scale' 100 0 ^100 200
```

sets the bottom left of the window to be row 0 column 0. The top left of the window is row 100 column 0. The bottom right of the window is row 0 column 200. The direction of the vertical axis has been reversed to be upwards.

Pen

Syntax: Control.Draw 'Pen' Style [[Color] Size]

Sets the Pen for further drawing commands.

The Style parameter is one of 0=Transparent, 1=Solid, 2=Insideframe, 3=Dash, 4=Dot, 5=DashDot, 6=DashDotDot. Default 1. *Note: Under MacOS, dashed or dotted lines do not display consistently unless they are either horizontal or vertical.*

The Color parameter can either be a single integer being the RGB values encoded 256+Blue Green Red where each color value is in the range 0 to 255, or separate Red Green Blue values in the range 0 to 255. Default 0 0 0 (Black)

The Size parameter is the pen width. (Default 1). If it is greater than 1, the Style must be 1.

Brush

Syntax 1: `Control.Draw 'Brush' Style [Color]`

Syntax 2: `Control.Draw 'Brush' ^2 FileName`

Sets the Brush for further drawing commands.

In the first form, the `style` parameter is one of 0=Transparent, 1=Solid, 2=Horizontal grid pattern, 3=Vertical grid pattern, 4=Forward diagonal, 5=Backward diagonal, 6=Cross pattern, 7=Diagonal cross patterns. Default 1.

The `color` parameter can either be a single integer being the RGB values encoded 2561Blue Green Red where each color value is in the range 0 to 255, or separate Red Green Blue values in the range 0 to 255. Default 0 0 0 (Black)

You can also set the `style` parameter to ^2, as shown in the second form above. In this case it should be followed by the name of a bitmap file, which will be used as the background fill pattern. (Note: APLX automatically caches this bitmap for you and releases it when it is no longer needed. There is no need for the APL programmer to obtain a bitmap handle and free it explicitly).

Color

Syntax: `Control.Draw 'Color' Color`

Sets the foreground (and optionally background) color for further text drawing commands.

The `color` parameter can either be a single integer being the RGB values encoded 2561Blue Green Red where each color value is in the range 0 to 255, or separate Red Green Blue values in the range 0 to 255. The default is 0 0 0 (Black).

If you want to specify the background color as well, you can provide two encoded integers, or a length 6 vector of two sets of RGB values. In this case the text foreground color is the first color you specify, and the background color is the second. Note that the background color will not be used if the current text mode is transparent - see the `Mode` command.

Font

Syntax: `Control.Draw 'Font' Name [Size] Style [Charset] Angle]`

Sets the font for further text drawing commands. The parameters are the same as for the normal `font` property, with the addition of an optional drawing `Angle`. This is expressed in units of one tenth of a degree, anticlockwise from the horizontal. (This parameter is ignored under MacOS 8 and 9.) If any parameter is an empty vector, it is left unchanged.

Mode

Syntax: `Control.Draw 'Mode' Opaque [[Mode] PolyType]`

Sets the drawing mode for further drawing and text commands.

The `Opaque` parameter is 0 for transparent text and 1 for opaque text, or `-1` to indicate 'leave unchanged'. The default is transparent. If you set `Opaque` to 1, text will be drawn with the background color selected using the `Color` command.

The `Mode` parameter is an integer in the range 1 to 16, or `-1` to indicate 'leave unchanged'. It determines how pixels drawn using the `Brush` or `Pen` interact with the existing pixels in the window. The exact meaning of the values depends on the system on which APLX is running; they are mapped so that the most useful ones give rise to similar effects under MacOS, Windows and Linux. The possible values and the resulting pixels which appear on the window are:

Value	Effect under Windows/Linux	Effect under MacOS
1	Pixels are replaced with Black	Pixels are replaced with Black
2	Inverse of the logical OR of the drawn pixel color and existing pixel color	<i>Not implemented</i>
3	Logical AND of the existing pixel color and inverse of the drawn pixel color	<i>Not implemented</i>
4	Inverse of drawn pixel color	Where the pen or brush pattern is 1, the existing pixel is unchanged. Where it is 0, the pen or brush color is used
5	Logical AND of the drawn pixel color and inverse of the existing pixel color	<i>Not implemented</i>
6	Inverts the current pixel color	<i>Not implemented</i>
7	Logical XOR of the drawn pixel color and existing pixel color	Where the pen or brush pattern is 1, the existing pixel is inverted. Where it is 0, it is unchanged
8	Inverse of the logical AND of the drawn pixel color and existing pixel color	<i>Not implemented</i>
9	Logical AND of the drawn pixel color and existing pixel color	<i>Not implemented</i>
10	Inverse of the XOR of the drawn pixel color and existing pixel color	Where the pen or brush pattern is 1, existing pixel is unchanged. Where it is 0, it is inverted
11	Existing pixels are unchanged	Existing pixels are unchanged
12	Logical OR of existing pixel color and inverse of drawn pixel color	<i>Not implemented</i>
13 (default)	Pixels are replaced with drawn pixel color	Pixels are replaced with drawn pixel color
14	Logical OR of drawn pixel color and inverse of existing pixel color	<i>Not implemented</i>

15	Logical OR of drawn pixel color and existing pixel color	<i>Not implemented</i>
16	Pixels are replaced with White	Pixels are replaced with White

In practice, the values you are likely to find useful are:

- 13 (*default*): Replace the destination pixel with the drawn pixel
- 7: Draw using the XOR operation, which is reversible. Draw again, and the figure will be removed. This is useful for 'rubber banding' or temporary outlines of shapes.

Under Windows, the `PolyType` parameter determines the algorithm used for filling of complex polygon shapes. If it is 0, the 'alternate' algorithm is used. If 1, the 'Winding' algorithm applies. It is ignored under MacOS and Linux.

Draw method: Rendering commands

Clear

Syntax: `Control.Draw 'Clear'`

Erases the entire drawing area and fills it with the current Brush. (For a Printer object, it ejects the page instead). Note that this command does not delete saved commands associated with the control, so for efficiency and to avoid wasting memory you should not call this method repeatedly if the `autoRedraw` property is 1. Alternatively you should use the `Reset` keyword to delete saved commands before calling `Clear`.

Bitmap (from file)

Syntax: `Control.Draw 'Bitmap' Filename [[[Row Col] Height Width] Stretch]`

Alternative keywords: **'Picture'** or **'Icon'**

Draws an image from a file on the control. The `Filename` parameter is the path to the file containing the image. Under Windows, this can be a Bitmap (.bmp), Windows metafile (.wmf), Icon (.ico) or JPEG (.jpg or .jpeg) image. Under MacOS, the supported formats include bitmaps and JPEGs, as well as MacOS PICT files and usually other formats such as TIFF and GIF. Under Linux, the supported types depend on the operating system version but will typically include .png .xpm .jpg .jpeg .ico and .bmp files. (To display all formats, your Windows or MacOS system may need an appropriate graphics converter or QuickTime module installed). APLX automatically caches the picture in memory, and releases it when it is no longer needed. If you display multiple copies of the same picture, only one copy will be loaded in memory.

The optional `Row` and `Col` parameters determine the top left corner on the control where the picture will be displayed.

Usually the picture is displayed at its natural size. The optional `Height` and `Width` parameters allow you to specify the size of the rectangle in which it is displayed. If the `Stretch` parameter is 0 or omitted, the picture is clipped to this size. If it is 1, the picture is stretched or shrunk to fit the specified height and width. (However, for Icons this parameter must be 0).

You can also draw multiple bitmaps with a single command by passing a nested vector of multiple `Filename...Stretch` parameters.

Bitmap (from APL variable)

Syntax: `Control.Draw 'Bitmap' Array [[[Row Col] Height Width] Stretch]`

Alternative keywords: **'Picture'** or **'Icon'**

This second form of the `Bitmap` command allows you to pass an APL array which contains the picture as a set of RGB color values. The `Array` parameter is a two-dimensional integer array. Each element contains a color value for the corresponding pixel. The color value is encoded as 256⍠Blue Green Red where each color value is in the range 0 to 255.

The other parameters are as described above.

You can also draw multiple bitmaps with a single command by passing a nested vector of multiple `Array...Stretch` parameters.

Circle

Syntax: `Control.Draw 'Circle' Row Col Radius [][[Aspect] StartAngle EndAngle] Border] TiltAngle]`

Draws a circle, ellipse, arc, or pie slice. The figure is drawn with the current `Pen` and filled with the current `Brush`.

The `Row` and `Col` parameters determine the center of the circle or ellipse. The `Radius` parameter determines the radius of the circle or ellipse. If the `Aspect` parameter is 1 (default), the figure drawn will be a circle provided the current `scale` has the same units along both axes. If the `Aspect` parameter is greater than 0, the `Radius` parameter determines the radius along the horizontal axis, and the radius along the vertical axis is obtained by multiplying `Radius` by `Aspect`. If `Aspect` is less than 0, the `Radius` parameter determines the radius along the vertical axis and the radius along the horizontal axis is obtained by multiplying `Radius` by the absolute value of `Aspect`.

The optional `StartAngle` and `EndAngle` parameters are used for pie slices and arcs. They are measured in degrees, anti-clockwise from the right horizontal axis. The default is 0 360, i.e. a full circle or ellipse.

The `Border` parameter determines which lines are drawn with the `Pen`. It is the sum of:

- 1 Draw the circumference
- 2 Draw the radius at the starting angle
- 4 Draw the radius at the end angle

The default is 1. If you don't want any line to be drawn, you can set `Border` to 0 or use a transparent `Pen`. If you don't want the figure to be filled, use a transparent `Brush`.

The final optional parameter, `TiltAngle`, allows you to specify that you want the figure rotated from the horizontal. It is expressed in degrees counter-clockwise from the right horizontal axis. The rotation is applied after all of the other parameters have been evaluated, so the radius along what would have been the horizontal axis becomes the radius along the line at `TiltAngle` degrees from the horizontal. *(If you specify this parameter, drawing may be slower and the figure less precise than it would be if you omit it).*

You can also draw multiple figures either by passing multiple enclosed vectors of 3 to 8 parameters, or by passing a matrix of 3 to 8 columns.

Chord

Syntax: Control.Draw 'Chord' Row Col Radius [[[[Aspect] StartAngle EndAngle] Border] TiltAngle]

Draws a chord (ie the intersection of a circle and a line). The figure is drawn with the current Pen and filled with the current Brush.

The parameters are the same as for Circle, except for the Border parameter, which is the sum of:

- 1 Draw the circumference
- 2 Draw the intersecting line

You can also draw multiple chords either by passing multiple enclosed vectors of 3 to 8 parameters, or by passing a matrix of 3 to 8 columns.

Ellipse

Syntax: Control.Draw 'Ellipse' Row Col Height Width

Draws a circle or ellipse. The figure is drawn with the current Pen and filled with the current Brush.

This is similar to the Circle command, but only complete circles/ellipses can be drawn and the size of the figure is expressed in a different way. The Row and Col parameters determine the top, left of the rectangle which just contains the figure. The Height and Width parameters determine the size of the containing rectangle.

You can draw multiple figures by passing a nested vector of 4-element parameters, or by passing an N by 4 matrix.

Line

Syntax: Control.Draw 'Line' Points

Draws one or more connected lines using the current Pen.

The Points parameter is an N by 2 numeric matrix describing the points to be connected, where N is at least 2. The first column represents the Rows, and the second column the Columns.

You can also draw multiple series of lines in one command by passing multiple sets of Points arrays.

Seg

Syntax: Control.Draw 'Seg' Array [Array, Array...]

Draws one or more sets of connected lines (without polygon fill) in one operation, using the current Pen.

The argument comprises one or more M by N matrices of: Y1 X1 Y2 X2... Each row of the matrix corresponds to one set of connected lines. Points are taken in pairs from each row, so N must be even, and the sets of lines in each array must have the same number of segments.

This method is provided mainly for compatibility with APL+Win. You can achieve the same effect in a more flexible manner by supplying a nested argument to the `Line` keyword.

Poly

Syntax: `Control.Draw 'Poly' Points`

Draws a polygon using the current Pen, and fills the polygons using the current Brush.

The `Points` parameter is an N by 2 numeric matrix describing the points to be connected, where N is at least 3. The first column represents the Rows, and the second column the Columns. APLX automatically draws the last line to connect the last point to the first. For complex shapes, the `Mode` command specifies the algorithm used to determine which regions of the polygon are filled.

If you don't want the polygon to be filled, use a transparent brush.

You can also draw multiple polygons in one command by passing multiple sets of Points arrays.

Rect

Syntax: `Control.Draw 'Rect' Row Col Height Width [RoundingV RoundingH]`

Draws a rectangle or round-cornered rectangle. The rectangle is drawn with the current Pen and filled with the current Brush.

The `Row` and `Col` parameters determine the top, left of the rectangle, and the `Height` and `Width` parameters determine its size. For a round-cornered rectangle, the `RoundingV` and `RoundingH` parameters determine the diameter of the corners in the vertical and horizontal directions respectively.

You can also draw multiple rectangles in one command by passing a nested vector of 4- or 6-element parameters, or by passing an N by 4 or 6 matrix.

Text

Syntax: `Control.Draw 'Text' String Row Col`

Draws a text string in the current Font.

The `String` parameter is a character vector or matrix containing the text to be written. It can contain embedded carriage return (`␣`) characters. If it is a matrix, carriage returns are automatically inserted at the end of each line. The `Row` and `Col` parameters determine the starting position. Text can be written at an angle by setting the angle using the `Font` command.

TextSize

Syntax: R ← Control.Draw 'TextSize' String [String, String..]

Alternative keyword: '?Text'

Determines the size required to render one or more text strings in the current Font.

The argument comprises one or more character scalars, vectors or matrices. If only one string is supplied, the result is a 2-element numeric vector containing the height and width required to draw the string with the currently-selected font, in the current Draw scale. If more than one string is supplied, the result is a nested vector with one element per string. Each element is a two-element vector giving the height and width required for the corresponding string.

Draw method: Grouping and Control commands

Reset

Syntax: `Control.Draw 'Reset'`

Deletes all saved Draw command sequences associated with the control, and resets the Font, Pen and Brush to default values. The default Pen is a black solid line of 1 pixel width. The default Brush is the window background color. The default Font is Arial 10 point.

State

Syntax: `R ← Control.Draw 'State'`

Returns a nested array of the Draw commands which are associated with a control. This is the series of commands which APLX automatically uses to redraw the control when a window update is needed. You can use this to redraw the same image on a different control by passing it directly as the nested argument of the `Draw` method. This is particularly useful for printing the contents of a window; simply open a Printer object and pass the array to the `Draw` method of the printer.

GetBitmap

Syntax: `R ← Control.Draw 'GetBitmap'`

Returns an APL integer matrix, where each element contains an RGB color for the corresponding pixel. The color value is encoded as `256⍲Blue Green Red`, where each color value is in the range 0 to 255

GetSVG

Syntax: `R ← Control.Draw 'GetSVG'`

Returns a character vector containing a representation of the drawing in Scalable Vector Graphics (SVG) format. This can be useful for creating high quality resolution-independent drawings suitable for publication.

Copy

Syntax: `Control.Draw 'Copy'`

Places a copy of the drawing on the clipboard in the native format for the host

Group

Syntax: Control.Draw 'Group' ID

Sets the Drawing group for further commands.

All drawing commands for a particular control belong to a group. (The default is group 0). Using this command, you can cause a particular series of commands to be given a different group ID. The group ID must be a non-negative integer. Subsequently, you can either temporarily enable and disable the group (for example, to hide a particular part of the drawing), or delete the group altogether. You can use this feature for animation effects.

DisableGroup

Syntax: Control.Draw 'DisableGroup' ID

Temporarily disables one or more Drawing groups for a particular control. The control is redrawn without the commands in the disabled group(s) being executed. The ID parameter comprises one or more integers containing the groups to be added to the list of disabled groups.

EnableGroup

Syntax: Control.Draw 'EnableGroup' ID

Re-enables one or more Drawing groups for a particular control. The control is redrawn. The ID parameter comprises one or more integers containing the groups to be removed from the list of disabled groups.

DeleteGroup

Syntax: Control.Draw 'DeleteGroup' ID

Removes one or more Drawing groups from the list of saved commands for a particular control. The commands are removed entirely and the memory associated with them is released. The control is redrawn.

Section 7. Using the Chart and Series Objects

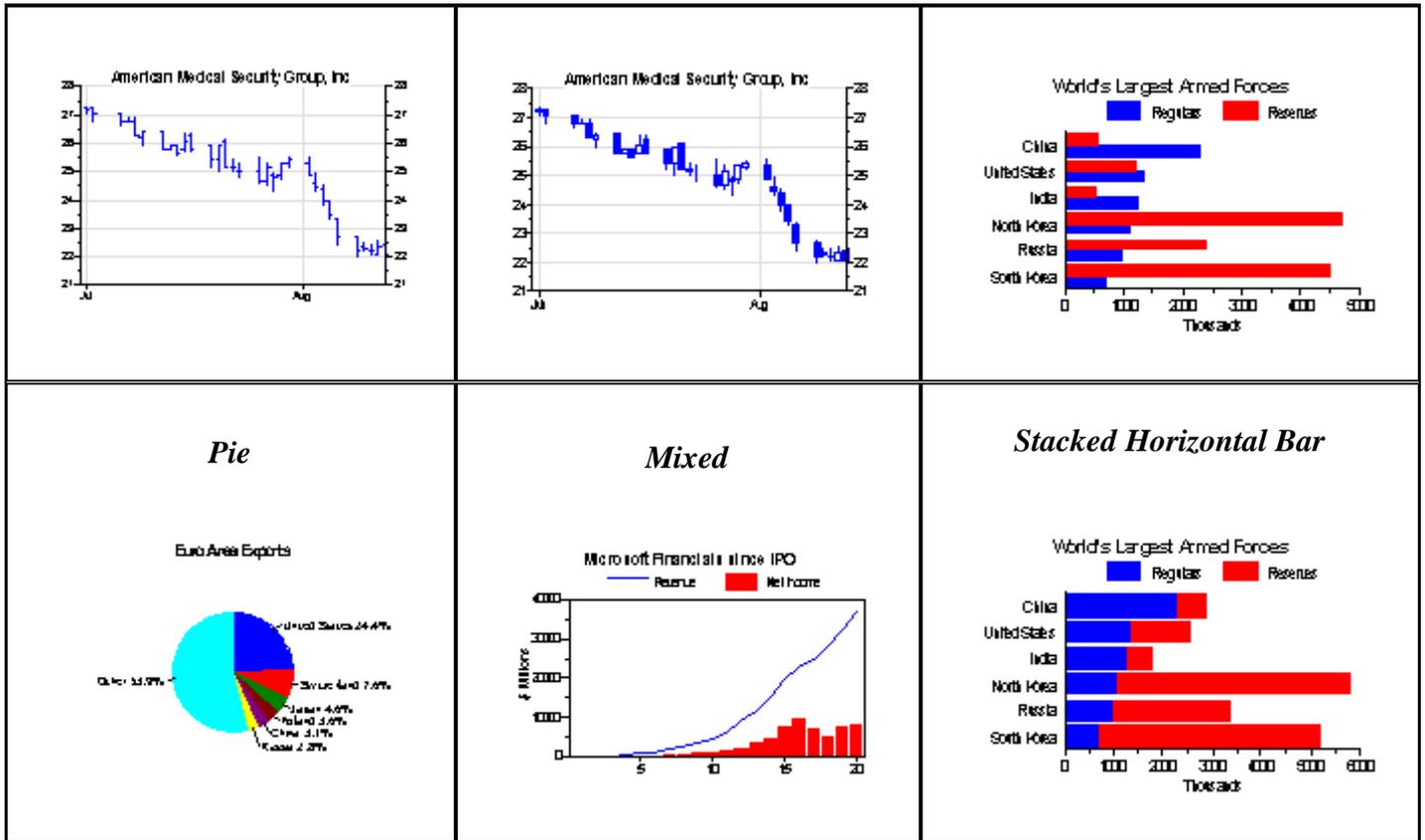
Chart and Series Objects: Introduction

The `Chart` class is a control which can display a range of different business, statistical and scientific graphs. The actual data which is displayed in the graph is held in one or more `Series` objects. (See the workspace `10 SAMPLESCHART` for examples).

Note: Using the `Chart` and `Series` objects, you have full control over the appearance of the graph. However, if you do not need this, a simpler (but less flexible) way of drawing a graph is to use the `CHART` system function, or simply to right-click on the name of a suitable variable and select 'Display as Chart' from the pop-up menu.

The `Chart` object can display the following types of graph:

<p style="text-align: center;"><i>Line</i></p> <p style="text-align: center;">Microsoft Financials (Logarithmic Scale) FY1985-2004</p> <p style="text-align: center;">Source: Microsoft Inc</p>	<p style="text-align: center;"><i>Scatter</i></p> <p style="text-align: center;">Recovery and Decay of Thorium Radioactivity: Rubenford and Boddy, 1902</p> <p style="text-align: center;">Source: Philosophical Magazine, 3(200), 302</p>	<p style="text-align: center;"><i>Bar</i></p> <p style="text-align: center;">World's Largest Armed Forces</p> <p style="text-align: center;">Source: CIA World Factbook</p>
<p style="text-align: center;"><i>Area</i></p> <p style="text-align: center;">Microsoft Financials FY1985-2004</p> <p style="text-align: center;">Source: Microsoft, Inc</p>	<p style="text-align: center;"><i>Stair</i></p> <p style="text-align: center;">UK Interest Rates May 1997 - September 2004</p>	<p style="text-align: center;"><i>Stacked Bar</i></p> <p style="text-align: center;">World's Largest Armed Forces</p> <p style="text-align: center;">Source: CIA World Factbook</p>
<p style="text-align: center;"><i>Hi Lo Open Close</i></p>	<p style="text-align: center;"><i>Candlestick</i></p>	<p style="text-align: center;"><i>Horizontal Bar</i></p>



Although you can customize many aspects of the display of a Chart object, it is designed so that it automatically selects defaults which should give a good appearance for the chart with minimal programming. For example, if a chart is re-sized, by default the font sizes are adjusted appropriately, and reasonable values for the chart scale, tick positions, and labels are automatically chosen.

Charts by default are displayed in color. You can select your own colors for all the major elements of a chart, or you can leave the Chart object to select defaults. You can also ask for the chart to be displayed in monochrome.

As well as being displayed within your windows, charts can also be printed, copied to the clipboard, and saved to file in various formats.

You can also customize a chart by adding your own graphics elements using the Draw method.

Except where noted below, the Chart object behaves identically on all of the APLX desktop platforms (Windows, MacOS, and Linux).

Using the Chart Object

The main steps you need to carry out to create a chart are as follows:

1. Create a Chart object, typically setting the chart type using the `type` property, and as appropriate setting labels for the chart title, axis labels, etc.
2. Create one or more Series objects, as children of the Chart object, and use these to specify the data which the chart will display. You can specify the data in various ways, typically using the `values` property, or the `xvalues` and `yvalues` properties separately.
3. Optionally, customize the chart's appearance by setting various properties of the Chart object. You can also set various properties of the child Series objects (for example, to set the colors used to display each series).

The Chart object responds dynamically to each change you make. (You can switch this off using the `update` property). So, having created a chart, you can for example convert a line chart to an area chart by changing the `type` property.

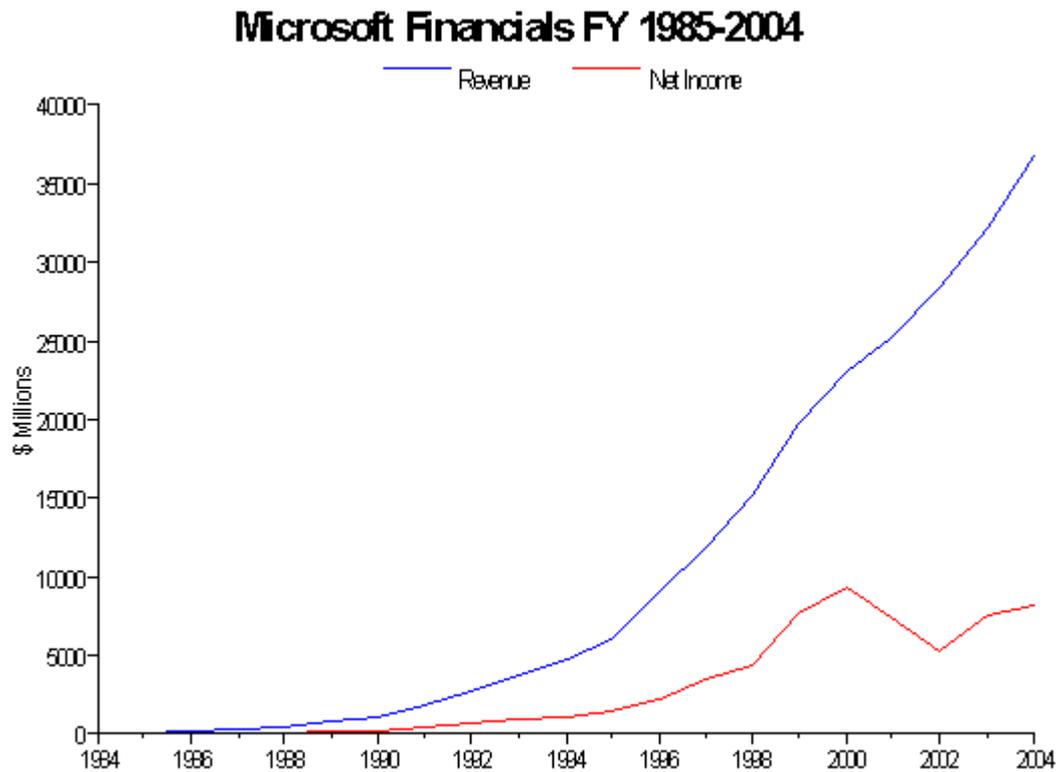
Here is an example of a function which produces a simple chart showing two different series:

```

    vChart_Microsoft;MS;data;years;revenue;income;ΠIO
[1]  a Sample line chart
[2]  MS←'□' ΠNEW 'Window' ♦ MS.title←'Sample chart'
[3]  MS.Chart.New 'Chart' ♦ MS.Chart.align←1 ♦ MS.Chart.type←'line'
[4]  MS.Chart.title←'Microsoft Financials FY 1985-2004'
[5]  MS.Chart.yaxislabel←'$ Millions'
[6]  a
[7]  MS.Chart.s1.New 'series' ♦ MS.Chart.s1.caption←'Revenue'
[8]  MS.Chart.s2.New 'series' ♦ MS.Chart.s2.caption←'Net Income'
[9]  a
[10] a Set up data
[11] ΠIO←0
[12] years←1985+i20
[13] revenue←140 198 346 591 805 1186 1847 2777 3786 4714 6075
[14] revenue←revenue,9050 11936 15262 19747 22956 25296 28365 32187 36835
[15] income←24 39 72 124 171 279 463 708 953 1146 1453
[16] income←income,2195 3454 4490 7785 9421 7346 5355 7531 8168
[17] a
[18] MS.Chart.s1.yvalues←revenue ♦ MS.Chart.s1.xvalues←years
[19] MS.Chart.s2.yvalues←income ♦ MS.Chart.s2.xvalues←years
    v

```

This produces a window which contains the following chart:

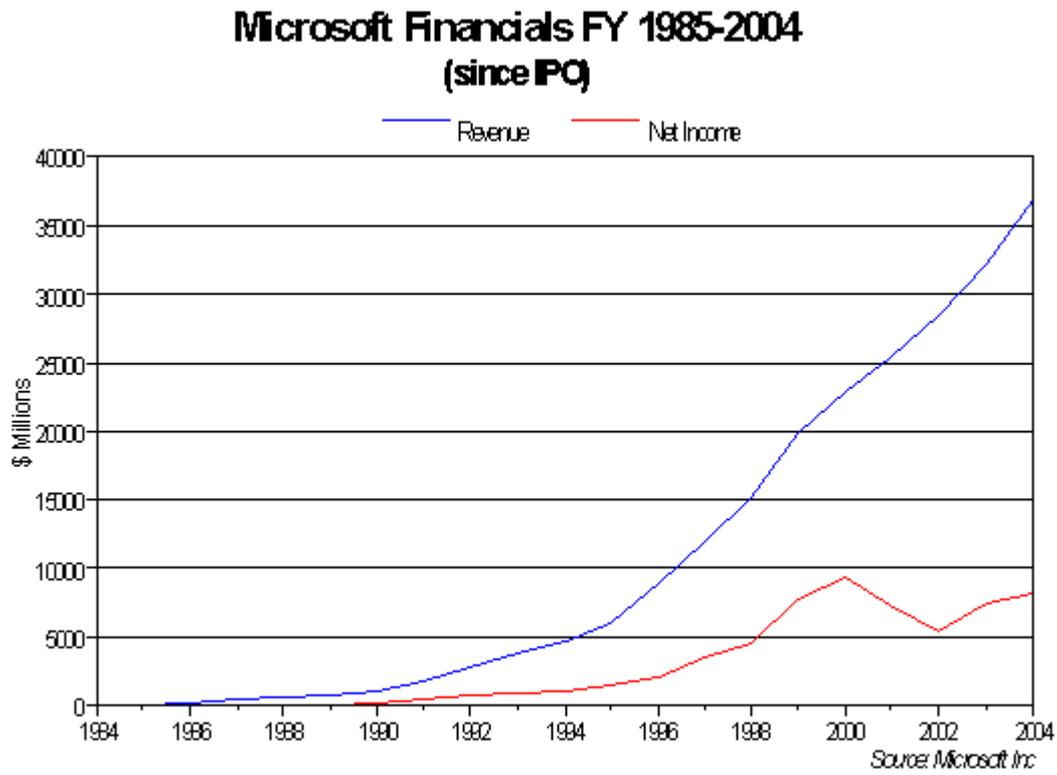


(Note that the number of ticks and labels shown may vary as the window is resized).

We can now customize the Chart object in various ways. For example, we can use the `style` property to display a grid, the `subtitle` property to add a subtitle, and the `note` property to add a note:

```
MS.Chart.subtitle< '(since IPO)'  
MS.Chart.note< 'Source: Microsoft Inc'  
MS.Chart.style< (1+8)
```

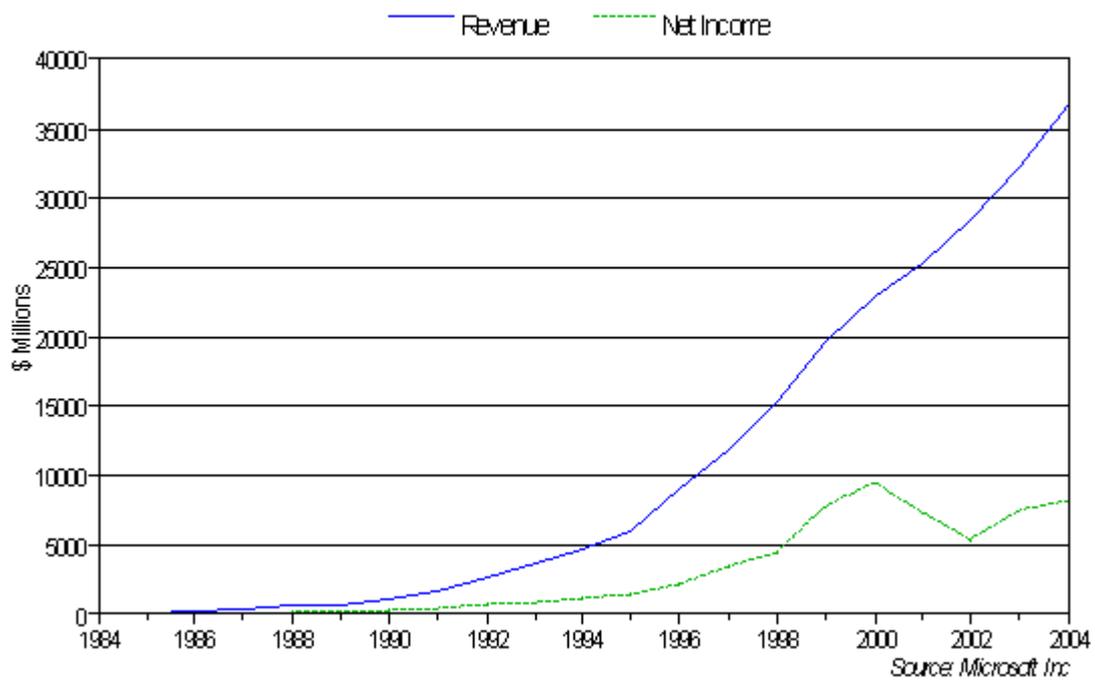
The graph will now look like this:



We can also set attributes of the individual series. For example, we might want the 'Net Income' line to be displayed as a dotted line, in green:

```
MS.Chart.s2.linetype<2
MS.Chart.s2.color<0 180 0
```

Microsoft Financials FY 1985-2004 (since IPO)



Axes, Coordinates and Scales

Apart from Pie charts, all the different chart types plot a dependent variable (known as Y) against another variable (known as X). By convention, the X axis is along the horizontal, and the Y axis is along the vertical, except for Horizontal Bar and Stacked Horizontal Bar charts, where the two are reversed.

By default, the Chart Object will select the *scale* of the chart (i.e. the range from the minimum X and Y coordinates to the maximum X and Y coordinates) automatically. It will also automatically choose where to place tick marks and labels along the axes. There are two types of tick mark; *major* tick marks (which are by default labelled), and *minor* tick marks (which are not labelled). The major tick marks are slightly longer. The Chart object chooses the intervals between tick marks to give a reasonable number of ticks depending on the size of the chart, and will place the marks at intervals which mean that the labels are round numbers (multiples of 1, 2, 5 or 10 times integral powers of ten).

The Chart object will also choose the *intercepts*, i.e. the point where the axes cross each other. This will usually be 0 if 0 is within the scale, else the lowest X or Y point on the scale. This means that the Y axis is usually shown on the left of the chart, but you can change this by using an alternate Y scale (see below), and you can also cause the Y axis to be drawn on both sides of the chart using the `style` property.

For Bar and Stacked Bar charts (and the Horizontal versions of these), the bars are by default drawn centred on integers 1, 2, 3.. *etc.*

The Alternate Scale

As well as the main Y scale, you can also specify that an *alternate* Y scale is used for one or more of the series on the chart, by setting the `usealtscale` property of the Series object. The alternate Y scale is shown on the right side of the chart, and has its own scale, ticks, and intercept. This is useful if you are plotting two different types of quantity on the same chart, or if the ranges of numbers of two series are very different. For example, in this chart, the left axis refers to cigarette consumption, and the right scale to obesity:

```

▽Chart_ObesityTobacco;□□I0;obesity;tobacco;Health
[1]  a Sample line chart with two different scales
[2]  Health←'□' □NEW 'Window' ◇ Health.scale←5 ◇ Health.size←450 600
[3]  Health.title←'Sample: Obesity and Tobacco'
[4]  Health.Chart.New 'Chart' ◇ Health.Chart.align←¯1
[5]  Health.Chart.type←'line' ◇ Health.Chart.style←1
[6]  Health.Chart.title←'Tobacco Consumption and Obesity'
[7]  Health.Chart.subtitle←'United Kingdom'
[8]  □□I0←0
[9]  a Set up the data
[10] obesity←1980 7 1991 14 1992 14 1993 15 1994 16 1995 16 1996 17
[11] obesity←13 2pobesity,1997 18 1998 19 1999 20 2000 21 2001 22 2002 22
[12] tobacco←1960 51.5 1970 49.5 1980 39 1990 30 1992 28
[13] tobacco←10 2ptobacco,1994 27 1996 28 1998 27 2000 27 2001 27
[14] Health.Chart.s1.New 'series'

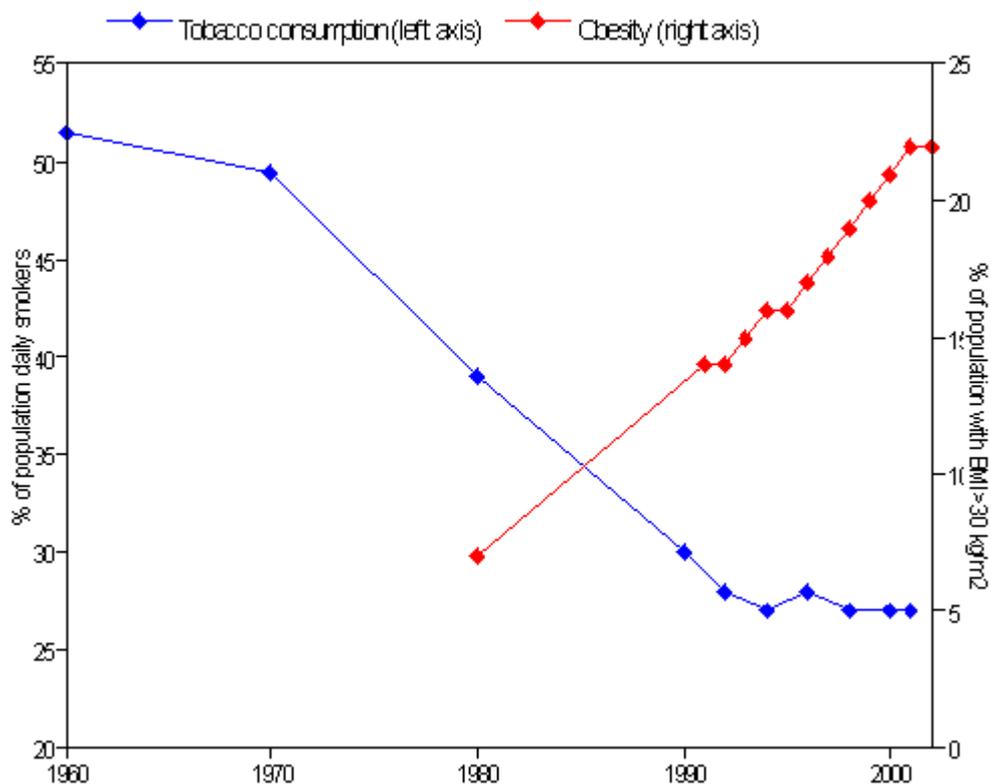
```

```

[15] Health.Chart.s1.marker<'◇' ◇ Health.Chart.s1.fillmarker<1
[16] Health.Chart.s1.caption<'Tobacco consumption (left axis)'
[17] Health.Chart.s1.values<tobacco
[18] Health.Chart.s2.New 'series'
[19] Health.Chart.s2.marker<'◇' ◇ Health.Chart.s2.fillmarker<1
[20] Health.Chart.s2.caption<'Obesity (right axis)'
[21] Health.Chart.s2.usealtscale 1
[22] Health.Chart.s2.values<obesity
[23] Health.Chart.yaxislabel<'% of population daily smokers'
[24] Health.Chart.yaltaxislabel<'% of population with BMI>30 kg/m2'
[25] Health.Chart.xmajorticks<1960 1970 1980 1990 2000
[26] ␣
[27] ␣ Wait until window is closed
[28] 0 0␣WE Health
      ▾

```

Line [15] of this function causes the second series to be drawn using the alternate scale, which is unrelated to the main Y axis scale. The effect is as follows:



Specifying your own tick positions and intercepts

If you wish, you can choose your own positions for the tick marks along any of the axes. You specify the values where major tick marks should be drawn by writing a vector of values to the Chart properties `xmajorticks`, `ymajorticks`, or `yaltmajorticks`. (The previous chart shows an example of this on line [19].). Similarly, you can select where minor tick marks should be drawn by writing to the `xminorticks`, `yminorticks`, or `yaltminorticks` properties.

If you specify your own tick positions, the final scale chosen will normally be from the lowest tick mark you specify to the highest. However, the scale will be extended if any of the points in any enabled Series is outside this range, so that the scale can represent all points on all the series.

In addition, you can change where the axes cross by specifying the `xintercept`, `xaltintercept`, and `yintercept` properties.

Specifying your own tick labels

By default, the Chart object writes a label next to each major tick mark, showing the numeric value of the point on the axis. If you wish, you can specify your own labels using the properties `xlabels`, `ylabels`, or `yaltlabels`. You can write to these properties as a text matrix, carriage-return delimited text vector, or as a nested vector of character strings. However, except for bar charts and pie charts, your labels will be displayed only if you also specify the major tick positions along the same axis (this is because otherwise the chart object would not know where you want the labels).

There are two special cases. For Bar charts (including the stacked and horizontal variants), there is no need to specify the X tick positions, since the bars are always drawn at 1, 2, 3... For Pie charts, there are no tick positions as such, but the `xlabels` strings (if any) will be used to label each pie.

Specifying axis labels

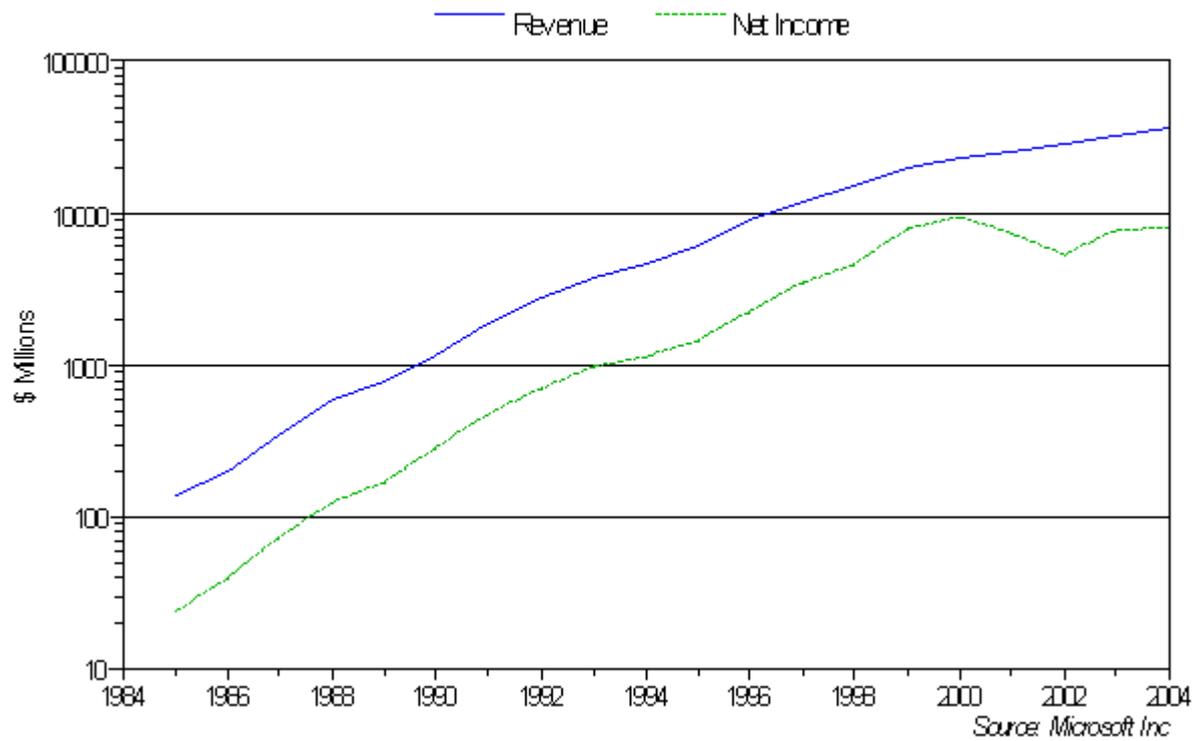
The `xaxislabel`, `yaxislabel` and `yaltaxislabel` properties allow you to specify a label for the axis as a whole (such as '\$ Millions' or 'Kg/m'). This is not the same as the tick labels, which label individual ticks along the axis.

Logarithmic Scales

Sometimes the range of data contained in a series is so large that a logarithmic scale is more appropriate. You can cause a logarithmic scale to be used for an axis by setting one of the `xlogscale`, `ylogscale` or `yaltlogscale` properties of the Chart to 1. (This works only if a logarithmic scale is valid, i.e. the values must all be greater than 0). For example, in the chart of the Microsoft financial results shown previously, the values for the early years do not show clearly because they are so tiny compared with the later values. A logarithmic scale shows the year-on-year growth in a more even fashion:

```
MS.Chart.ylogscale←1
```

Microsoft Financials FY 1985-2004 (since IPO)



Note that, in this example, the minor ticks represent 2, 3, .. 9 times the power of ten shown at the previous major tick.

Chart Object Properties

Setting the chart type:

type

The `type` property determines the type of chart. It is a text string, which is one of the following values (case is ignored when you assign the string):

`'line'` (*default*)

Selects a **Line Chart**. Each series is drawn as lines between successive X, Y points. By default, markers are not shown at each point. (If you want markers, set the `marker` property for the series). You can specify the color and line type for each series individually, or leave the Chart object to select a default.

`'stair'`

Selects a **Stair Chart**. Same as a Line chart, except that instead of the line being drawn directly from each X,Y point to the next, a horizontal line is drawn first to the next X position, and then a vertical line is drawn to the next Y position.

This sort of chart is useful for displaying series where the Y values change in step-wise fashion (for example, a chart of interest rates set by a central bank against time, or a chart showing a digital representation of an analog value).

`'area'`

Selects an **Area Chart**. Similar to a line chart, except that each series is drawn as a filled area between the lines connecting the X,Y points, and the X axis. The series are drawn in the order in which they were created, so a later series may conceal some or all of an earlier series. You can specify the color and fill pattern for each series individually, or leave the Chart object to select a default. (*Tip: If you want to avoid concealing one series underneath another, you can set the `fillpattern` property of each series to be something other than `solid`.*)

`'scatter'`

Selects a **Scatter Chart**. No line is drawn between the X,Y points of the series; instead, a marker is drawn at each X,Y point. You can leave the Chart object to choose a default marker type, or specify it yourself using the `marker` property of each series.

This sort of chart is useful for displaying experimental results. (Sometimes, you will also want to include a separate line series showing a theoretical fit to the experimental results, or a least-squares regression to show a trend. You can do this using the `mixed` chart type).

`'bar'`

Selects a **Bar Chart**. The graph is drawn as a series of filled vertical bars from the X axis. (The X values are ignored, except possibly to set labels for the bars). If you have more than one series, the bars for each series are displayed side-by-side. As with the other chart types, you can set the fill color and/or pattern individually for each series, or leave the Chart object to choose defaults. You can also set the thickness of each bar (relative to the available space) using the `barwidth` property.

`'stackedbar'`

Selects a **Stacked Bar Chart**. Like a bar chart, but the bars for each series are placed on top of each other at each data point, rather than side-by-side. This type of chart is useful for showing both the total value of a quantity (for example, total sales by quarter), and the contributions to the total (for example, how much each region contributed to the total).

Normally, stacked bar charts are used only for series where all the Y data values are positive. If there are negative Y values encountered at a given X value, there will in effect be two stacked bars drawn; the positive ones will be stacked on top of each other in the positive direction, and the negative ones will be stacked on top of each other in the negative direction.

`'horizbar'`

Selects a **Horizontal Bar Chart**. Same as a Bar chart, except that the bars are drawn horizontally.

`'horizstackedbar'`

Selects a **Horizontal stacked bar chart**. Same as a Stacked Bar chart, except that the bars are drawn horizontally.

`'hilo'`

Selects a **High-Low-Open-Close (HLOC) chart**. Unlike the previous chart types, for an HLOC chart you need to specify more than one Y value for each X value for a series. As a minimum, you should specify the High and Low values (using the properties `highvalues` and `lowvalues`). For a financial chart, these typically represent the highest price and the lowest price reached by a stock or commodity during a particular trading period. The chart is drawn as a series of vertical bars between the High and Low values for each X value (thus showing the price range for the period). You can optionally specify Open values for the series (using the property `openvalues`), typically representing the opening price for the period. These are drawn as a small tick to the left of the vertical bar. You can also optionally specify Close values for the series (using

the property `closevalues`, which is a synonym for `yvalues`). These are drawn as a small tick to the right of the vertical bar.

Although primarily used for financial data, this type of chart can also be used for other applications, such as displaying experimental results. In this case, the Open and Close values would both be specified to represent the primary experimental results, with the High and Low values representing the confidence limits.

'candle'

Selects a **Candlestick chart**. This is a variant of the High-Low-Open-Close chart. Again, a line is drawn between the High and Low values, but the difference between the Open and Close values is shown by a rectangle. If the Open value is greater than the Close, the rectangle is filled with the series foreground color, otherwise it is filled with the background color.

This type of chart is often used to show price trends over time; the filled rectangles make it very obvious whether the closing prices were higher or lower than the opening prices for each trading period.

'pie'

Selects a **Pie chart**. Draws one or more circular areas, where each value is shown as a slice of the circle representing the proportion of the total attributable to each series. As such, they are rather like stacked bar charts, except that you cannot tell the total value at each point, only the proportion contributed by each series. Negative values are ignored. If there are multiple pie charts displayed, they are automatically arranged in rows and columns to make the best use of the available space. Each pie will have the same radius.

By default, the pie chart will be accompanied by a 'legend' which shows the colors used for each series (provided you have specified a `caption` property for the series). However, the pie chart often looks better if you disable the legend (by setting the `placelegend` property of the Chart object to 'none'), and use the `style` property to cause each pie slice to be labelled with the series caption and a percentage value.

Caution: The Chart object displays one pie chart for each X value, with each series contributing a slice to each pie (unless the value is zero or negative). It does NOT display one pie per series. Thus, in the limiting case where you want a single pie showing four segments, you need to create four separate series, each with just one value. Although this may seem somewhat unnatural at first, it is implemented in this way for consistency with other chart types (especially bars and stacked bars). This means you can swap between the different chart types to display the same data in different ways. This approach also allows you to set the color, fill pattern and caption for each segment by setting properties of the Series objects. The QUICKPIE function in the workspace 10 SAMPLESChart can be used if you want to create a simple pie chart from an APL array.

You can label the individual pies by setting `xlabels` property of the Chart object.

`'mixed'`

Selects a **Mixed** chart type, with different types for each Series. For all of the Chart types above, each series in the chart is displayed in the same way. Sometimes, however, you might want to mix different types of chart together. For example, you might want to show experimental results as a Scatter chart, and the prediction of a mathematical model of the data as a Line chart. Or you might want to show monthly profit figures as a Bar chart, and the cumulative year-to-date profit as a Line chart.

If you set the `type` property of the Chart object to `'mixed'`, then each series will be drawn according to the `type` property of the individual Series object. This can be any of the above types except for horizontal or stacked bars, or pies.

Setting titles and labels:

title *or* **caption**
subtitle
note

These properties are all text vectors, and are simply labels displayed around the chart. Normally, the `title` property is displayed in the largest font, and is intended to represent the title for the whole chart. The `subtitle` property is displayed below the title, in a slightly smaller font, and is intended as a sub-title for the whole chart. The `note` property is typically displayed in a small font, and is intended to represent additional information such as a copyright notice or acknowledgement. However, you can use these labels in any way you wish, or omit them altogether by leaving them as empty vectors.

xaxislabel
yaxislabel
yaltaxislabel

These properties are also text vectors, and are labels displayed adjacent to the axes. They are usually used to indicate the units or meaning of the scale along the axis.

Specifying where labels and legends are drawn:

placetitle
placenote
placelegend

These properties determine where various labels of the chart are drawn.

The `placetitle` property determines where the `title` and `subtitle` (if any) are drawn. It is a text string, and can be one of the values `'top'`, `'topleft'`, `'topright'`, `'bottom'`, `'bottomleft'`, `'bottomright'`. The default is `'top'`, meaning the title is centered above the chart.

The `placenote` property determines where the `note` (if any) is drawn. It can have the same values as the `placetitle` property. The default is `'bottomright'`.

The `placelegend` property determines where the chart legend is drawn. The legend shows, for each series which has a non-empty `caption` property, a sample line, marker, or filled rectangle as appropriate, next to the caption. (If a given series has no caption, it will not appear in the legend). This property is a text string, which can be one of the values `'top'`, `'left'`, `'right'`, `'none'`, or an empty vector. If it is `'none'` or an empty vector, the legend is not shown. The default is `'top'`.

Choosing fonts:

font

The `font` property determines the base font used for the Chart object. By default, the Chart object uses this base font for every text element it draws, with the size being chosen as a proportion of the window size, and different sizes for the different elements (for example, the `title` is drawn as the biggest). By changing the `font` property, you can change the font family used to display all elements unless they are specifically altered using one of the specific font properties described below. The `font` property uses the same form as the `font` property of other controls, i.e. it is a nested vector of (Font name) (Size) (Style) (Character set). However, the `Size` parameter is not used. The default is Arial, plain text.

Note: Take care when changing this property to use a font which displays well at a wide range of sizes, unless the chart is not resizable. We recommend that you use a TrueType or OpenType font in all cases.

fontaxis

fontlegend

fontnote

fonttitle

These properties allow you to specify the font for individual elements of the chart. Any changes you make to these override the setting of the base `font` property.

These properties are specified in a similar way to ordinary `font` properties, as a nested vector of (Font name) (Size) (Style) (Character set). However, the `Size` is defined in a special way. If it is set to a value between 0 and 1, it means that the height of the font should be the specified proportion of the window size, defined as the smaller of the window height and width. Values in the region of 0.03 to 0.05 are reasonable. This is recommended for resizable charts, to ensure that the labels remain in proportion to the chart as a whole. (If the font gets too small, a lower limit is imposed). If the font size is specified as an integer greater than 1, it is interpreted

as a font size in Points, and will be fixed irrespective of the size of the Chart. This is likely to mean that the chart does not display well at very small or large sizes.

If the Font name is an empty vector, the base font family is used. If the Size or Style element is `-1`, the corresponding font attribute is left unchanged.

Specifying ticks, tick labels, and intercepts:

xmajorticks
ymajorticks
yaltmajorticks
xminorticks
yminorticks
yaltminorticks

By default, the Chart object chooses a suitable scale automatically, and chooses sensible major and minor tick positions. It labels the major tick positions with the corresponding numeric value.

These properties allow you to specify your own positions for the major or minor ticks along the X axis, Y axis, and alternate Y axis (if used). Each property is a numeric vector, specifying where each tick should be drawn. If both the major and minor tick vectors for a particular axis are empty (which is the default), the Chart object will position the tick marks automatically. If either the major or minor tick vector is not empty, ticks will be drawn only where you have specified them.

If you specify your own tick positions, the scale for a particular axis will usually range from the lowest tick mark you specify to the highest. However, it will be extended to include any data point which would otherwise fall outside the range.

xlabels
ylabels
yaltlabels

These properties can be used to change the labels written next to the major tick marks. By default, the Chart object labels the major ticks with the corresponding numeric value. If you specify one of these properties, then your labels will be used instead. However, (except for the X axis of a bar chart or pie chart), you must also specify the corresponding major tick positions, otherwise the Chart object will not know where the labels should be placed.

The labels can be specified either as a character vector with embedded carriage returns, or as a text matrix, or as a nested vector of character vectors. The first label is used for the first major tick position, and so on. If there are more labels specified than tick positions, the surplus labels are ignored. If there are fewer, the ticks which do not have user-defined labels are labelled with the default numeric value.

xlogscale
ylogscale
yaltlogscale

These are Boolean scalar properties. If set to 0 (which is the default), a normal linear scale applies. If set to 1, a logarithmic scale is used for the corresponding axis.

You cannot set a logarithmic scale for an axis where any data point is 0 or negative. If this situation occurs, the scale will revert to linear.

xintercept
yintercept

These properties can be used to specify where the X axis and main Y axis should cross the other axis. By default the X axis is placed at the bottom edge of the graph, or at the Y value of 0 if the Y axis ranges from negative to positive values, but you can specify a different position by changing the `yintercept` property. Similarly, the Y axis is by default placed at the left edge of the graph, or at X value 0 if the X axis ranges from negative to positive values, but you can specify a different value using the `xintercept` property.

xaltintercept

Returns the X position where the alternate Y axis (if any) crosses the X axis. Because the alternate Y axis is always drawn on the right-hand edge of the chart, this property is read-only.

xscale
yscale
yaltyscale

These are read-only properties, each comprising a two-element numeric vector. They return the limits of the X, Y and Alternate Y axes respectively.

Changing the appearance of the chart:

style

The `style` property of a Chart object is the sum of a set of flags:

- 1 Draw box around graph drawing area (i.e. the axes or individual pie charts)
- 2 Draw grid lines on X major ticks
- 4 Draw grid lines on X minor ticks
- 8 Draw grid lines on X major ticks
- 16 Draw grid lines on Y minor ticks

64 Draw the Y axis on the right as well as left

128 Label pie slices with the series name

256 Label pie slices with the percentage

512 Label pie slices with the actual value

The default is 0.

border

The `border` property is a Boolean scalar. If it is set to 1, a border is drawn around the whole Chart object, including the titles and other labels.

monochrome

Normally, charts are drawn in color, with different series being drawn in different colors to distinguish them. Sometimes, however, you may want the chart to be drawn in monochrome, for example if you are printing it on a black-and-white printer. The `monochrome` property allows this. If you set it to 1, all the elements of the chart will be drawn in the Chart object's Foreground color (by default, black). Since this means that the different series cannot be distinguished by color, they will instead be distinguished by line type (solid, dashed, dotted, etc) or by fill pattern (vertical hatch, horizontal hatch etc). See the `linetype` and `fillpattern` properties of the Series object for more information on these.

color or colour

The `color` property is defined in the same way as for other objects. It sets the Foreground and Background color of the Chart as a whole. Colors can be specified in one of two ways. In the first way, you specify a vector of three integers (Red, Green and Blue), so that for example 255 0 0 is pure red and 255 255 255 is white. In the second way, you specify a single integer which encodes the three values in base 256 as $256 \times \text{Blue} + \text{Green} + \text{Red}$. In addition, three special values are recognised; -1 means use the parent's foreground or background color as appropriate; -2 means use the default color (black for foreground and white for background); -3 (valid only for the Background) means that the background of the Chart is transparent, allowing you to draw the chart on top of a picture.

When the Chart is drawn, the control is first set to the Background color (unless it is transparent), and then the various graphics elements are drawn. By default, the axes, grids,

labels and titles are all drawn in the Foreground color, but you can change this using more specific properties (see below). The data lines, markers, bars, or pie slices for each series are drawn in the color associated with the series (or in the Foreground color if the `monochrome` property is set to 1). Note that the Chart object chooses a default color for each series when it is created; you can change this by setting the `color` property for the child Series object.

coloraxis *or* **colouraxis**
colorgrid *or* **colourgrid**
colorlegend *or* **colourlegend**
colornote *or* **colournote**
colortitle *or* **colourtitle**

These properties allow you to set specific colors for the various individual elements of the chart. They are specified as a single color value (vector of three separate RGB values, or a single integer encoding the three values). The default is `-1`, meaning use the Chart object's foreground color. These properties are ignored if the `monochrome` property is set to 1.

linewidth

The `linewidth` property of the Chart is an integer scalar, which specifies the width in pixels of lines drawn in the chart. The default is 1. The value set here is used for all lines (axes, grids, and the data series lines), unless you specify a different value for a particular element using one of the more specific properties described below. You can set the line width for a particular data series by using the `linewidth` property of the child Series object.

axiswidth
gridwidth

These two properties allow you to specify the line width (in pixels) for the axis and grid lines respectively, as an integer scalar. The default is `-1`, which means use the value set in the Chart object's `linewidth` property.

barwidth

The `barwidth` property allow you to specify the width of the bars used in bar charts. It is an integer scalar, in the range 0 to 100, representing the percentage of the available width used to draw the bars at each X value. (For a Bar or Horizontal Bar chart with multiple series where the bars for each series are placed side by side, this width is shared equally between the different series). If the `barwidth` property is set to 100, then 100% of the available space is used, so there will be no gaps between the bars. If it is set to a low value (say 20), then the bars will be very narrow, with lots of white space between them. The default is 80.

margin

This property sets the space which the Chart object leaves around the chart. (In addition to this, the Chart object automatically allows extra space for the titles, legend, axis labels, and tick labels.) It is a numeric vector of four elements, representing the Left, Right, Top and Bottom margins, in the units set by the standard `scale` property of the control (do not confuse this with the X and Y scales used for plotting data). The default is 16 pixels for all four margins.

When you use the `Print` method to print the chart, the values in this property are added to the print margins specified on the Printing tab of the APLX Preferences dialog.

Other properties:

workarea

This is a read-only property. It returns a vector of four numbers, representing the Top Left Bottom Right of the data area of the graph within the control (i.e. the area enclosed by the axes, excluding any margin, titles, legend, or axis labels). It is expressed in the units set by the standard `scale` property of the control. It can be used if you need to draw extra features on the chart (see the notes on the `Draw` method below).

update

Normally, any changes you make to the chart take effect almost immediately (when events are next processed). In some cases, this might give a flickering effect if the chart ends up being redrawn several times because you are making a lot of changes to it. You can temporarily disable redraws by setting the `update` property to 0, and then making a sequence of changes to the chart. When you set the `update` property back to 1, the chart will be redrawn once, with all of the changed properties.

bitmap

The `bitmap` property is read-only. It returns a numeric matrix, which represents the image of the chart as an array of pixel values, with each pixel encoded as the RGB value. The image can be displayed in a `Picture` object, or (using an `Image` object), manipulated further and saved to file in any of the supported formats (such as JPEG or GIF).

picture

The `picture` property is read-only. It is similar to the `bitmap` property, in that it represents the image of the chart. However, instead of representing each pixel of the chart it is currently drawn, it contains the series of drawing commands (encoded in a platform-specific way into a vector of integers). It is thus a vector-graphics version of the chart image, and can be re-drawn in good quality at different sizes, typically in a `Picture` object. You can also write it to file, or to the clipboard. (The `picture` property is not currently implemented under Linux).

svg

This read-only property returns the representation of the chart in Scalable Vector Graphics (SVG) format. This format is ideal for publishing your charts, since SVG will retain high-quality output on high-resolution devices.

Chart Object Methods

Print

This is a method without arguments. It causes the chart to be printed on the currently-selected printer. The chart will fill as much of the page size set using the APLX Page Setup dialog as it can, within the margins set in the Preferences dialog and any extra margins set using the chart's `margin` property, whilst keeping the same aspect ratio as the chart has on screen.

Copy

This is a method without arguments. It causes the chart to be copied to the clipboard as a picture. Under Windows, it is copied as a Windows metafile, so for example you can paste it into Microsoft Word as a resizable image. Under MacOS, it is copied as a PICT object. Under Linux it is copied as a bitmap.

Save *filename* [*type*]

This method causes the chart to be saved to a graphics file. The *filename* parameter is the full path of the file to be written. Alternatively, it can be an empty vector, in which case a standard file dialog is displayed to allow the user to choose the name. The optional *type* parameter can be one of the following:

0 : Write the file using the native vector graphics format of the host (*not supported under Linux*):

Under Windows the vector format used is the Windows Enhanced Metafile (`.emf`). Under Macintosh OS X the vector format used is a PICT file.

1 : Write the file in a bitmap format (*not supported under MacOS*).

The format used is a Windows-style bitmap file (`.bmp`).

2 : Write the file in Scalable Vector Graphics (SVG) format.

The default is 0 for Windows and Macintosh, 1 for Linux.

Note: If you want to save the chart in a different format, for example as a GIF, you can do this using an Image object. To do this, you need to read the `bitmap` property of the Chart, write this to the `bitmap` property of the Image, set the Image `format`, and then call the `Save` method of the Image.

Draw *keyword param1 param2..*

The `Draw` method is fully implemented for `Chart` objects, allowing you to customize the chart by adding your own graphic elements and annotations. These are displayed after the main chart has been drawn. You can also draw in the `X` and `Y` coordinate system of the chart. See the section on Customising Charts below.

Pointtochart *Y X***Pointtochartalt *Y X***

These two methods convert a point (Vertical and Horizontal values in the current `scale` of the control), to the coordinates of the chart (either the equivalent `Y` and `X` value of the main axis, or the Alternate `Y` and `X` axis). The argument is a two-element numeric vector of `Y`, `X`, and a two-element numeric vector is returned.

These two methods are particularly useful for responding the mouse-down events on the chart. For example, you might want to display an information panel if a point on the chart is clicked. If you set the `scale` property to 5 (pixels), these methods directly convert the pixel position on the control to the coordinates of the chart.

Chartalttopoint *Y X***Charttopoint *Y X***

These methods carry out the inverse transformation, from chart coordinates to a point on the control, expressed in the current `scale` of the control (you will probably want to set the `scale` property to 5 for pixels). They can be useful if you are adding your own annotations or other graphic elements using the `Draw` method.

Series Object Properties

Setting the data for the series:

values

The `values` property provides a quick way of specifying the X and Y values of a series in one operation. You can assign to it in the following ways (they are tested for in this order):

(a) *2 by N numeric matrix:*

In this case the first row is assumed to contain the X values, and the second row the Y values.

(b) *N by 2 numeric matrix:*

In this case the first column is assumed to contain the X values, and the second row the Y values. (Note: for a 2 by 2 matrix, case (a) applies).

(c) *Numeric scalar, vector, 1 by N matrix, or N by 1 matrix:*

In this case the data provided is assumed to represent the Y values of the series, and the X values are implicitly set to 0, 1, 2....N-1

If you read back this property, the data is always returned in format (a), i.e. as a 2 by N matrix of X values, Y Values

xvalues

yvalues

These properties can be written as numeric scalars, vectors or one-row matrices. They provide an alternative way of specifying the X and Y values respectively for points on the series. The `xvalues` property can be omitted (i.e. left as an empty vector), in which case they are assumed to be 0, 1, 2... up to the number of points specified in the `yvalues` property less 1. Otherwise, you should always provide the same number of values for X and Y.

If you read back one of these properties, the data is always returned as a numeric vector.

highvalues

lowvalues

openvalues

closevalues

These set the data used for High-Low-Open-Close and Candlestick charts. They are numeric vectors, each representing one of the four Y data points associated with each X value in the `xvalues` property. You should normally provide the same number of values for each of these properties, although `openvalues` and `closevalues` are optional.

Note that the `closevalues` property is actually a synonym for `yvalues`, and contains the same data.

Using the Alternate scale for the series:

usealtscale

This is a Boolean scalar. It defaults to 0, meaning that the series should be drawn using the main Y scale (i.e. the left axis). If you set it to 1, the series will instead be drawn using the alternate Y scale (i.e. the right axis).

If you want only one Y axis, but you want it to appear on the right instead of the left, set this property to 1 for all the series in the chart, so they all use the alternate axis. Alternatively, if you want only one axis, but you want it to be drawn on both the left and the right, leave this property as 0 for all the series in the chart (so they all use the main axis), and use the `style` property of the Chart object to specify that you want the main axis drawn on both sides (value 64).

Enabling/Disabling and Showing/Hiding the series:

enabled

This is a Boolean scalar. It defaults to 1, meaning that the series is enabled and should be included in the chart.

If you set this property to 0, the series is not included in the chart at all. This means that it is not displayed, and also that it takes no part in determining the scale used by the chart.

visible

This is also a Boolean scalar which defaults to 1, meaning that the series is visible (provided the `enabled` property is also 1).

If you set this property to 0, the series is not displayed when the chart is drawn. However, unless the `enabled` property is set to 0, the series *will* be taken into account when determining the scale of the chart.

This is useful for two purposes. Firstly, you can temporarily hide a series without the scale of the chart altering. Secondly, you can force the chart to extend the scale by including a dummy series which is enabled but not visible.

You can also use the `Show` and `Hide` methods to make the series visible or not; these have the same effect as changing the `visible` property.

Labelling the series:**caption or title**

This property is a text vector, which associates a label with the series. By default it is an empty vector, which means the series has no label. If you give it a label using this property, then it will appear in the legend for the chart, with a sample line, marker or filled rectangle to show which series is which on the chart. (The legend will not be shown if the Chart object's `placelegend` property is set to 'none' or to an empty vector.)

For a Pie chart, the `caption` can also be shown directly next to the Pie slice, if the Chart object's `style` property includes the value 128.

Setting the appearance of the series:**marker**

When a series is drawn as a Scatter chart, each point is denoted by a marker symbol. By default, the symbol to use is selected when the series is created. By changing the `marker` property, you can choose a particular marker symbol. In addition, if you specify a marker using this property, the marker will be also be displayed if the series is shown as a Line chart.

The marker is specified by assigning a character scalar, which is one of the following APL symbols: + × * ○ ϕ ⊖ ▽ Δ ◇ □ ↑ ↓ → ← It can also be a space or empty vector, in which case no marker is drawn.

(The actual marker is drawn is not a character, but a shape similar to that of the corresponding APL symbol. The APL font is not used to render the marker shape.)

fillmarker

Where a marker is drawn, by default it is drawn in outline only. If you set the `fillmarker` property to 1, the marker will be filled with the current marker background color (as set by the `colormarker` property), provided it is one of the following shapes: ○ ϕ ⊖ ▽ Δ ◇ □

color or colour

By default, the Chart object will choose a unique color for each Series object when it is created, but this property allows you to select a particular color. It is specified as a single color value (vector of three separate RGB values, or a single integer encoding the three values). It is used for drawing lines, and filling bars or areas, associated with the data of the series.

The `color` property is ignored if the Chart's `monochrome` property is set to 1.

colormarker or colourmarker

Usually, any marker associated with a series is drawn in the color set for the series as a whole (i.e. the `color` property, which is either chosen automatically or set explicitly). If you wish, you can display the marker in a different color by setting the `colormarker` property. You can also specify a background color for filling the marker. It is specified either as a single color value (vector of three separate RGB values, or a single integer encoding the three values), in which case it is used for both the foreground and background colors, or as two color values (length six vector of RGB values, or two integers encoding the color values), in which case the foreground color is used to outline the marker, and the background color is used to fill it if the `fillmarker` property is 1. The default values of `-2 -2` mean use the same color as is used for the series as a whole.

The `colormarker` property is ignored if the Chart's `monochrome` property is set to 1.

fillpattern

Normally, bars and pie slices associated with a series are filled with a solid block of color. If you wish, you can specify a different fill pattern for the series, by assigning one of the following values to the `fillpattern` property:

1=Solid, 2=Horizontal grid pattern, 3=Vertical grid pattern, 4=Forward diagonal, 5=Backward diagonal, 6=Cross pattern, 7=Diagonal cross pattern

If the Chart's `monochrome` property is set to 1, and you haven't specified your own fill pattern, a default pattern from one of the above will be assigned to the Series object to distinguish it from other series on the chart.

linetype

Normally, lines associated with a series are drawn as a continuous (solid) line. If you wish, you can specify a different line type for the series, by assigning one of the following values to the `linetype` property:

1=Solid, 3=Dash, 4=Dot, 5=DashDot, 6=DashDotDot

If the Chart's `monochrome` property is set to 1, and you haven't specified your own line type pattern, a default type from one of the above will be assigned to the Series object to distinguish it from other series on the chart.

linewidth

This property is an integer scalar, which determines the width (in pixels) of lines drawn for the series. By default, it is `-1`, which means that the width should be taken from the `linewidth` property of the parent Chart object. By assigning a different value, you can specify a different line width for this series only.

Setting the series type for Mixed charts:

type

Normally, each series on the chart is shown in the same way, as determined by the `type` property of the parent Chart object. For example, all the series are displayed as bars if the chart `type` is set to `'bar'`.

Sometimes, however, you may wish to display different series in different ways on the same chart. To do this, you need to set the `type` property of the parent Chart object to `'mixed'`. The way in which each series on the chart will be displayed will then depend on the `type` property of each individual Series object. You can set this to one of: `'line'` `'scatter'` `'area'` `'bar'` `'stair'` `'hilo'` `'candle'`. The default is `'line'`.

Caution: Be careful when mixing series displayed as bar charts with the other types. Bars are by default drawn centered at X coordinates 1, 2, 3..., if no `xvalues` are set. By contrast, other types of graph are drawn either with the X coordinates explicitly set using `xvalues`, or at X coordinates starting at 0 (*not* 1) if no X values are specified.

Customizing the Chart using the Draw method

You can add extra features to the display of a Chart object by using the `Draw` method. This works exactly as for other controls, allowing you to write text or add shapes such as polygons to the chart. When the Chart control is redrawn, it first renders the chart itself, and then runs your `Draw` method commands. (The same is true when the Chart object is printed, copied to the clipboard, or read back as a bitmap).

For this to be useful, you need to be able to relate your `Draw` commands to the chart itself. To assist in this, you can use the Chart object's `Chartalttopoint` `Charttopoint` `Pointtochart` `Pointtochartalt` methods, which allow you to convert between the X Y coordinates of the chart and the ordinary `scale` of the Chart control.

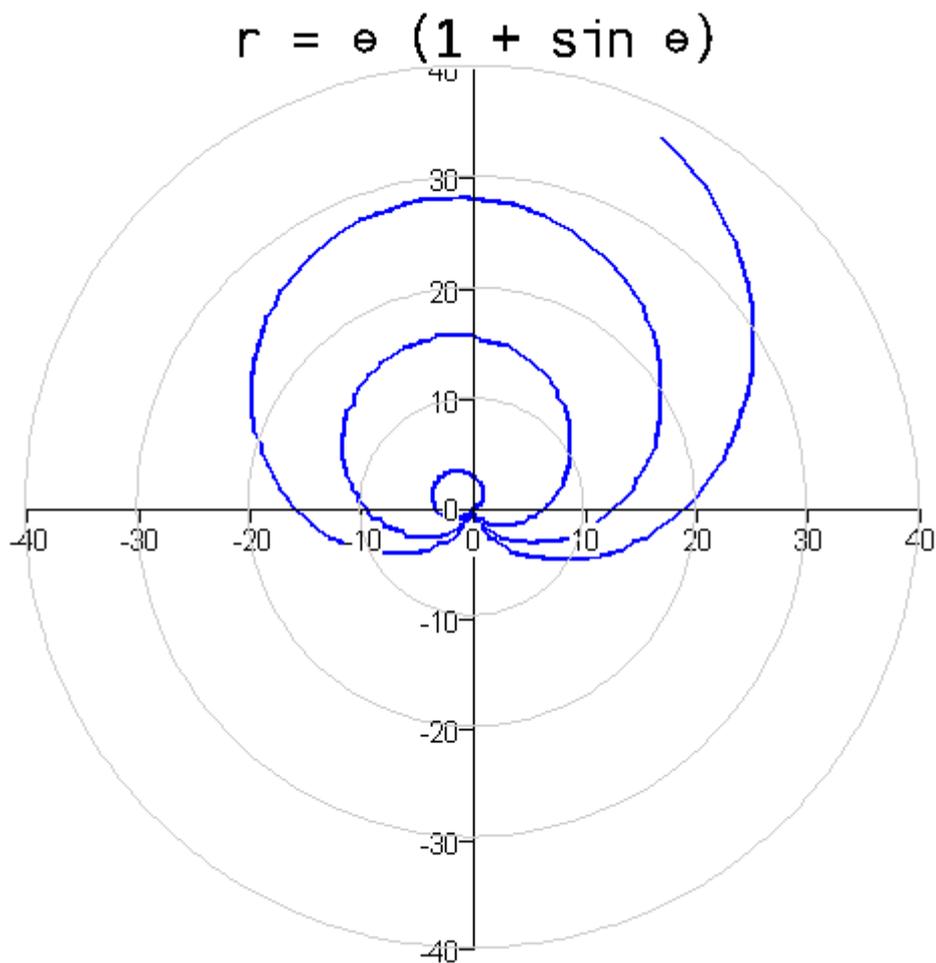
However, an easier way of ensuring that your `Draw` commands match up to the chart layout is to set the `Draw` command `scale` to 7, meaning 'use the Chart object's X and Y scales'. If you do this, all of your `Draw` commands will automatically be correctly rescaled if the chart is re-sized, as in this example where the `Draw` method is used to add circular coordinate lines to a polar chart (lines 26 to 30):

```

    vChart_Polar;xdata;ydata;angle;radius;PI0;Polar
[1]  a Sample line chart used as a 'polar' chart
[2]  a Draw line where radius is a function of angle from horizontal
[3]  Polar←'□' □NEW 'Window' ♦ Polar.scale←5 ♦ Polar.size←548 512
[4]  Polar.title←'Sample: Polar'
[5]  Polar.Chart.New 'Chart' ♦ Polar.Chart.align←~1
[6]  Polar.Chart.fonttitle←'APLX Upright'
[7]  Polar.Chart.title←'r = e (1 + sin e)'
[8]  PI0←0
[10] angle←(ι400)÷20
[11] radius←angle×(1+1◦angle)
[12]  a
[13]  a Convert polar coordinates to XY coordinates
[14]  xdata←radius×2◦angle
[15]  ydata←radius×1◦angle
[16]  a
[17]  a Set the scale to be ~40 to +40 along both X and Y axes
[18]  Polar.Chart.xmajorticks←~40 ~30 ~20 ~10 0 10 20 30 40
[19]  Polar.Chart.ymajorticks←~40 ~30 ~20 ~10 0 10 20 30 40
[20]  a
[21]  a Plot the data as though it were a line chart
[22]  Polar.Chart.s1.New 'series' ♦ Polar.Chart.s1.xvalues←xdata
[23]  Polar.Chart.s1.yvalues←ydata ♦ Polar.Chart.s1.linewidth←2
[24]  a
[25]  a Use the Draw method to add circular reference lines
[26]  Polar.Chart.Draw 'Scale' 7          a Use Chart coordinates
[27]  Polar.Chart.Draw 'Pen' 1 200 200 200 a Light gray
[28]  Polar.Chart.Draw 'Mode' ~1 9       a Mode 9 so that ticks show through
[29]  Polar.Chart.Draw 'Brush' 0         a Transparent so circle not filled
[30]  Polar.Chart.Draw(='Circle'),((=0 0),~10×1+ι4)
[31]  a
[32]  a Wait until window is closed
[33]  0 0p□WE Polar
    v

```

This give the following effect, where the circular gray lines come from the Draw method commands:



Section 8. OLE, OCX and ActiveX Programming

OCX/ActiveX Controls and OLE Automation

Note: This section applies to Windows only

APLX for Windows supports OLE Automation (also known as just Automation, or COM), a powerful facility which allows your APLX functions to use external software and controls to enhance the functionality of your applications. There are three ways in which you can use this feature:

- (a) You can place external OCX or ActiveX controls directly on your APLX windows. For example, you can use the Formula One Excel-style spreadsheet control as though it were a built-in APLX control. This is done simply by creating a window and then creating an external control on it.
- (b) You can embed OLE (Object Linking and Embedding) documents in your windows. For example, you could include a Microsoft Word document as part of a more complex window, merging the Word menus and toolbar with your own. To allow this, APLX includes a control called an 'OLEContainer' which you place on your window. You can then specify the document to be embedded or linked, or allow the user to select one.
- (c) You can invoke and exchange data and commands with external OLE Server Applications. This is somewhat similar to (b), but the application runs independently of APLX and is not embedded in one of your windows. For example, you could invoke Excel, cause it to load a spreadsheet, and extract the data from the spreadsheet as an APL nested array, without the user being involved.

In each case, you create an object using `NEW` or `WI`. This object will have properties, methods, and events which belong to the external software but which you can access just as you do with ordinary APLX objects. (To avoid any possible ambiguity, each external property, method, or event name is prefixed with an 'x', but you can omit this if you prefer.)

Properties supporting Automation

The `System` object has three properties which each return an N by 4 nested array of the external controls and servers installed on your system. These properties are:

```
xclasses OCX/ActiveX controls - case (a) above
oledotypes Document types which can be embedded in an OLEContainer - case (b) above
oleclasses Server applications which you can invoke - case (c) above
```

The returned array has one row for each external control or server installed (not all of these will necessarily be accessible).

The first column is the plain text name, for example 'Chart FX'. The second column is an alphanumeric string enclosed in curly brackets, which is the unique ID of the automation object. The third column is the object class name, which is usually a period-delimited string giving the vendor

name, object class name and optionally the version number, for example 'SoftwareFX.ChartFX.20'. (The fourth column is reserved for future use).

You can use any of these as the class name to identify the control to APLX, but we recommend that you use the third.

You can also use the Control Browser (on the Tools menu) to see the same information together with more detail on the properties, methods, events and constants exported by the control.

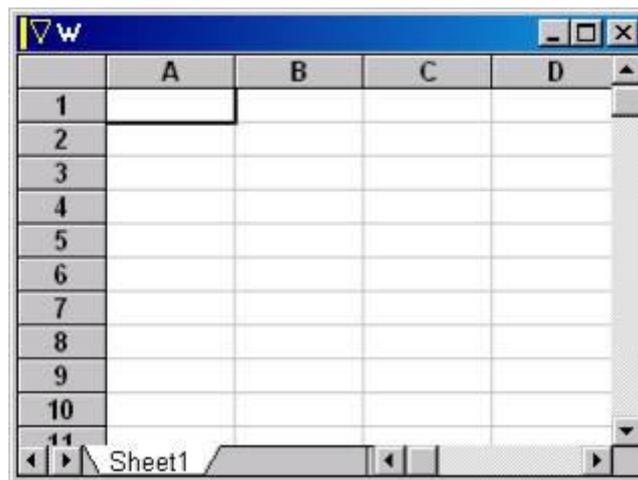
Using OCX/ActiveX Controls

An OCX or ActiveX control is an external control which you can place on one of your APLX windows. There are many such controls available from Microsoft and from third-party software vendors. Examples include controls for drawing graphs, spreadsheet controls, document viewers, calendar controls, etc.

The use of an ActiveX/OCX control is best illustrated by an example. Suppose you want to use the Formula One spreadsheet control from Visual Components, Inc. You must first create a window, and then place the control on it. The class name of the control is one of the names returned by the `xclasses` property (or you can use the Control Browser window). For Formula One, we can use the programmatic ID 'VCF1.VCF1Ctrl1.1':

```
W←'□' □NEW 'Window'
W.F1.New 'VCF1.VCF1Ctrl1.1' ♦ W.F1.align← 1
```

At this point, we have created a window and the control, ready for use but currently without any data:



Now we can see what properties and methods the external control exposes:

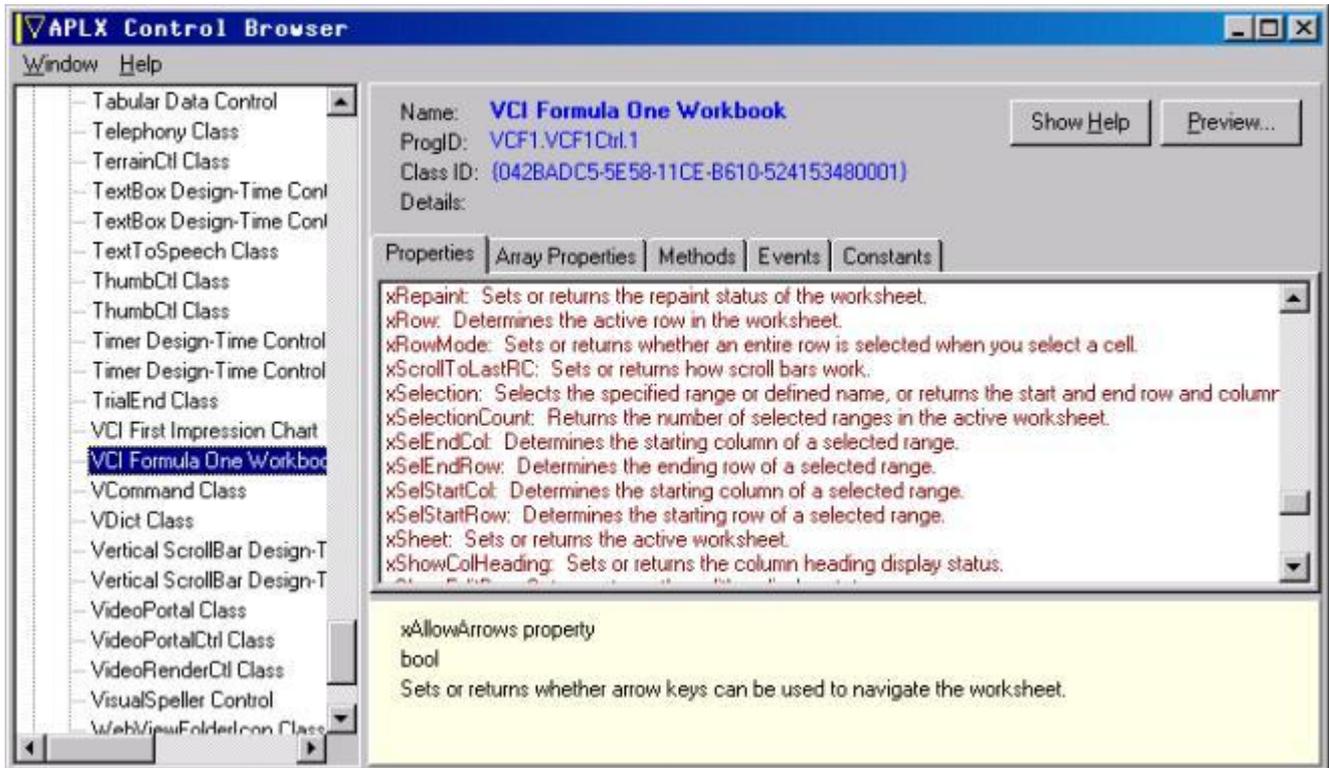
W.F1.properties

```
align anchors autodraw caption children class color data doublebuffered dragsou
rce droptarget enabled events handle methods name opened order pointer pro
perties scale self size sourceformats tabstop targetformats tie verbs visi
ble where winptr xAllowArrows xAllowAutoFill xAllowDelete xAllowDesigner x
AllowEditHeaders xAllowFillRange xAllowFormulas xAllowInCellEditing xAllow
MoveRange xAllowObjSelections xAllowResize xAllowSelections ... etc etc
```

W.F1.methods

```
Click Close Delete Doverb Draw Focus Hide New Open Paint Resize Send Set Show S
howaboutbox Showproperties Trigger XAboutBox XAddColPageBreak XAddPageBrea
k XAddRowPageBreak XAddSelection XAttach XAttachToSS XAutoFillItemCount X
CalculationDlg XCancelEdit XCheckRecalc XClearClipboard XClearRange XColor
PaletteDlg XColWidthDlg XCopyAll XCopyRange XCopyRangeEx ... etc etc
```

There are many more which we have not shown for brevity. The first few are standard APLX properties/methods, and then those beginning with 'x' are defined by the Formula One control. The APLX Control Browser gives you more details:



You can also look at the Formula One help file for full documentation.

Using external Properties and Methods

You can use the properties, and call methods, just as you would for built-in APLX controls:

Read the current row (Note: You can use 'row' or 'xrow' as the property name)

```
W.F1.row
1
```

Set the current row and column:

```
W.F1.row←2
W.F1.col←3
```

Set the current cell as text:

```
W.F1.text←'23.2'
```

Read it back as a number:

```
W.F1.number
23.2
1 + W.F1.number
24.2
```

Array properties

Some external controls use 'Array Properties', which are collections of properties indexed by one or more parameters (usually but not always integers). APLX uses a special syntax for these. See the separate description of Array Properties for details.

Responding to external Events

Many ActiveX/OCX controls trigger events, just like those associated with APLX built-in controls. You can cause APLX functions to be called when an event occurs, by setting the associated 'on..' property of the control. The names of the events are those defined by the control, with a leading 'X'. Thus, if the OCX control has an event called 'Recalculate', you assign your event handling function to the property `onXRecalculate`. (You can omit the 'X' if the name does not clash with an APLX built-in name). You can find out the names of the events in the usual way:

W.F1.events

```
onClick onClose onDb1Click onDestroy onDragDrop onDragEnd onDragEnter onDragLeave
onDragOver onDragStart onFocus onHide onKeyDown onKeyPress onKeyUp onMouseDown
onMouseMove onMouseUp onOpen onSend onShow onUnFocus onXCancelEdit
onXClick onXDb1Click onXEndEdit onXEndRecalc onXKeyDown onXKeyPress onXKeyUp
onXModified onXMouseDown onXMouseMove onXMouseUp onXObjClick onXObjDb1Click
onXObjGotFocus onXObjLostFocus onXObjValueChanged onXRClick onXRDb1Click
onXSelChange onXStartEdit onXStartRecalc onXTopLeftChanged onXValidationFailed
```

(Again the first few of these are the standard APLX events; the external control events begin with 'onX'). The Control Browser gives more information on parameters and data types.

Any parameters to the callback function can be retrieved using the `□WARG` system function.

For example, the Formula One spreadsheet-control allows you to specify a callback, to be triggered when the user clicks in a cell. The event name is `onXClick`, and the parameters are defined as:

```
void Click (long nRow, long nCol);
```

This means that it passes two integer parameters, which are the row and column of the cell in which the user has clicked.

You can associate an APL function with this event as follows:

```
W.F1.onXClick←'EVTCLICK'
```

(Note that in this case we must use the full name `onXClick`, because there is also an APLX built-in event called `onClick`).

In your APLX callback function you can retrieve the event parameters; `□WARG` will return a two-element vector with the row and column values:

```
    ▾EVTCLICK;□IO
[1]  □IO←1
[2]  'The user clicked in row ',(⊘□WARG[1]),' column ',⊘□WARG[2]
    ▾
```

Additional methods for OCX/ActiveX controls

There are some specific built-in APLX methods which apply to OCX controls. These are:

`Showproperties`: (No arguments). Displays the control's properties dialog, if there is one, allowing the user to set specific properties (for example, a charting control may allow the user to select the type of chart). It returns a boolean scalar 1 if the user pressed OK, or 0 if the user pressed Cancel.

`Showaboutbox`: (No arguments). Displays the control's About Box, typically giving copyright and version information.

See also the `verbs` property and the `Doverb` method.

Using OLE linked/embedded documents

The second type of OLE Automation supported by APLX is the embedding or linking of a document in one of your APLX windows, using a special control called an OLEContainer. ('Embedding' means that there is a copy of the document associated with the control, whereas 'linking' means that the control does not have a copy of the data but maintains a link to an existing file). The user can edit the document using the application associated with it.

As an example, suppose you want to embed a Microsoft Word document in your application. You must first create a window, and then place an OLEContainer control on it. We use the `align` property to cause it to fill the window:

```
W←'□' □NEW 'Window'
W.OLE.New 'OLEContainer'
W.OLE.align←1
```

Initially, there is no document in the container, and the properties and methods comprise only those which APLX defines:

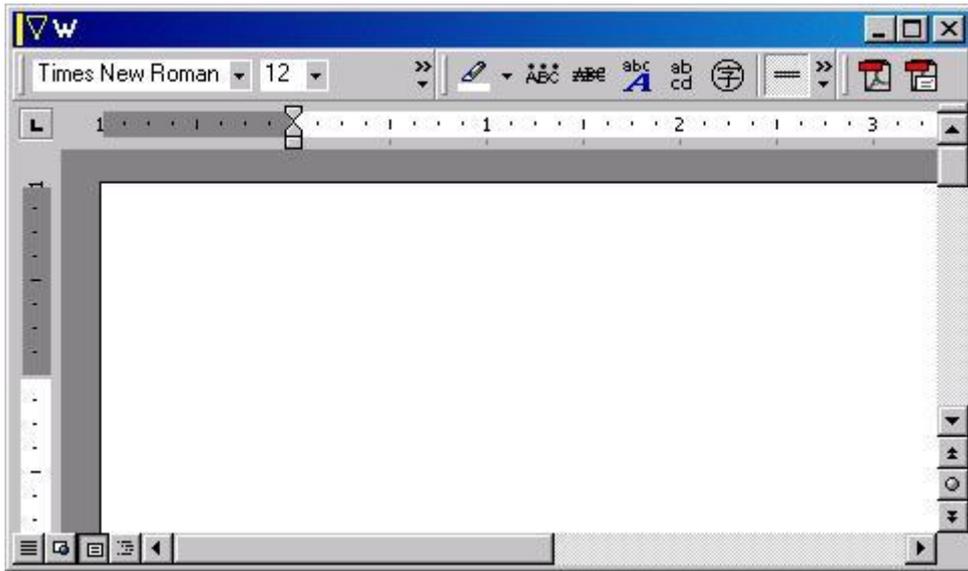
```
W.OLE.properties
align anchors autoactivate autodraw border canpaste caption children class color
contents data docstate doublebuffered dragsource droptarget enabled events
file handle methods modified name opened order pointer properties scale
self size sizemode sourceformats style tabstop targetformats tie verbs visible
where winptr

W.OLE.methods
Click Close Closedocument Delete Doverb Draw Focus Hide New Open Paint Refresh
Resize Send Set Show Showproperties Trigger
```

You can invoke Word to embed a document in several ways, as described below. These include writing a filename (or empty vector) to the `file` property, writing a class name to the `progid` property, or writing a previously saved document to the `contents` property. In this example, we create a new blank Word document:

```
W.OLE.progid←'Word.Document'
```

Our APLX window now looks like this, and the user can enter text (if there were a menu on the window, Word's menu would be merged with it):



If the server application exposes its own properties and methods, these will now be accessible to APLX. Not all OLE servers provide such facilities, but Word certainly does:

W.OLE.properties

```
align anchors autoactivate autodraw border canpaste caption children class color
contents data docstate doublebuffered dragsource droptarget enabled events
file handle methods modified name opened order pointer properties scale
self size sizemode sourceformats style tabstop targetformats tie verbs visible
where winptr x_CodeName xActiveTheme xActiveThemeDisplayName xActiveWindow
xActiveWritingStyle xApplication xAttachedTemplate xAutoHyphenation
xBackground xBookmarks xBuiltInDocumentProperties xCharacters xClickAndType
peParagraphStyle xCodeName xCommandBars xComments xCompatibility xConsecutiveHyphensLimit
xContainer xContent xCreator xCustomDocumentProperties xDefaultTabStop
xEmail xEmbedTrueTypeFonts xEndnotes xEnvelope ... etc etc
```

W.OLE.methods

```
Click Close Closedocument Delete Doverb Draw Focus Hide New Open Paint Refresh
Resize Send Set Show Showproperties Trigger XAcceptAllRevisions XActivate
XAddToFavorites XApplyTheme XAutoFormat XAutoSummarize XCheckConsistency XCheckGrammar
XCheckSpelling XClose XClosePrintPreview XCompare XComputeStatistics XConvertNumbersToText
XCopyStylesFromTemplate XCountNumberedItems XCreateLetterContent XDataForm XDetectLanguage
XEditionOptions ... etc etc
```

We have again truncated the lists for brevity.

Some properties are themselves objects. For example, the `xContent` property is actually a pointer to a Content object:

W.OLE.xContent

```
1826228
```

You can explore the properties and methods of the sub-object in the normal way, using a dot notation to drill down the levels:

W.OLE.xContent.properties

```
xApplication xBold xBoldBi xBookmarkID xBookmarks xBorders xCase xCells xCharacters
xCharacterWidth xColumns xCombineCharacters xComments xCreator xDisableCharacterSpaceGrid
xDuplicate xEmphasisMark xEnd xEndnotes xFields xFind
```

```
xFitTextWidth xFont xFootnotes xFormattedText xFormFields xFrames xGrammarChecked
xGrammaticalErrors xHighlightColorIndex xHorizontalInVertical xHyperlinks xID
xInformation xInlineShapes xIsEndOfRowMark xItalic xItalicBits xKana xLanguageDetected
xLanguageID xLanguageIDFarEast xLanguageIDOther xListFormat xListParagraphs
xNextStoryRange xNoProofing xOrientation xPageSetup xParagraphFormat xParagraphs
xParent xPreviousBookmarkID xReadabilityStatistics xRevisions xRows xScripts
xSections xSentences xShading xShapeRange xSpellingChecked xSpellingErrors
xStart xStoryLength xStoryType xStyle xSubdocuments xSynonymInfo xTables
xText xTextRetrievalMode xTopLevelTables xTwoLinesInOne xUnderline xWords
```

This allows you to set or retrieve the properties of the sub-objects. For example, to write text to the Word document:

```
W.OLE.xContent.text<-'Now is the time'
```

OLEContainer properties

The main OLEContainer properties are:

style: The style is the sum of a set of flags:

- 1 = Linked instead of embedded document (must be set before setting the 'file' property)
- 2 = Allow in-place activation
- 4 = Show as icon instead of document (must be set before setting 'file' property)

progid: A character vector determining the class of object (for example, whether it is a Word document or a Pinnacle Chart). It is of the same form as the third column of the System object's `oledoctype`s property. If there is a document loaded, reading the `progid` property allows you to find out what the document type is. Writing to the `progid` property creates a new empty document of the appropriate class. If you write an empty vector, a dialog is displayed allowing the user to choose.

file: A character vector determining the name of the document contained in the OLEContainer. If the OLE container is linked to a document, reading the `file` property allows you to find out its name. Writing to the `file` property causes the container to attempt to load the document (or link to it, if the `style` is 1). If you write an empty vector, a dialog is displayed allowing the user to choose a document.

autoactivate: Determines how the activation of the document takes place. One of:

- 0 = Manual (use the `DoVerb` method to activate the document)
- 1 = Automatic (APLX activates it automatically) [Default]
- 2 = Activate when the control gets focus
- 3 = Activate when the user double-clicks in the control.

sizemode: Determines how the document fits in the container. One of:

- 0 = Displays the OLE object at its normal size, clipping any parts that don't fit within the container. [Default]
- 1 = Displays the OLE object at its normal size, centering it within the container.
- 2 = Scales or shrinks the view of the OLE object to fit within the container, by scaling width and height proportionally.

- 3 = Scales or shrinks the view of the OLE object to fill the OLE container, without regard to preserving the proportions of the OLE object.
- 4 = Displays the OLE object at its normal size and automatically resizes the container to fit the size of the OLE object.

`docstate`: Read-only property giving information about the OLE document state. One of:

- 0 = There is no OLE object in the container.
- 1 = There is an OLE object in the container, but its server application isn't currently running.
- 2 = The OLE object's server is running.
- 3 = The OLE object is open in a separate window.
- 4 = The OLE object is activated inplace, but hasn't yet merged its menus or toolbars.
- 5 = The OLE object is activated inplace and menus and toolbars have been merged

`modified`: Boolean property, saying whether the document has been modified. You can reset this to 0 (false).

`verbs`: Read-only property. It returns a nested array of the application-specific 'verbs' which the server application defines. See the `Doverb` method for details.

`contents`: When you read the `contents` property, APLX returns a character vector containing the internal representation of the document (or the link) in the container, encoded as a series of bytes and including information about the type of the document. You should never change the contents of this vector, but you can store it in a file and write the same vector back to an `OLEContainer` to cause it to re-display the same document.

OLEContainer methods

The main `OLEContainer` methods are:

`Showproperties`: (*No arguments*). Displays the document's properties dialog.

`Doverb`: Takes an integer argument which is the ID of a 'verb' to perform. This can either be a positive integer representing one of the application-specific verbs (see the `verbs` property), or one of the standard verbs:

- 0 Specifies the action that occurs when an end-user double-clicks the object in its container.
- 1 Instructs an object to show itself for editing or viewing.
- 2 Instructs an object to open itself for editing in a window separate from that of its container.
- 3 Causes an object to remove its user interface from the view. Applies only to objects that are activated in-place.
- 4 Activates an object in place, along with its full set of user-interface tools, including menus, If the object does not support in-place activation, gives domain error
- 5 Activates an object in place without displaying tools, such as menus and toolbars, that end-users need to change the behavior or appearance of the object.
- 6 Used to tell objects to discard any undo state that they may be maintaining without deactivating the object.

`Refresh`: (*No arguments*). Causes the display to be updated to reflect any changes to the source document.

`Closedocument`: (*No arguments*). Closes the document, remembers its state, and disconnects from the server application. You can still read the `contents` property to retrieve the document contents.

Using OLE (COM) Server Applications

The third use of OLE automation is to link to an external server application. This is somewhat similar to embedding a document in an OLEContainer, but the application runs independently of APLX and is not associated with one of your windows. It may not even be visible. For example, you could invoke Word, cause it to load a document, and extract the text from the document, without the user being involved.

Using an OLE (also known as COM) Server Application is very similar to the previous examples, except that you create the object as a top-level object, using the external class name (as returned by the `oleclasses` property) as the APLX class name.

For example, this function invokes Excel and retrieves a spreadsheet as a nested array (see the workspace `10 SAMPLESEXCEL` for more examples):

```

    ▽R←GetSpreadsheet SHEETNAME;ROWS;COLS;COLNAMES;ΠIO
[1]  A
[2]  A APLX for Windows OLE Example:
[3]  A     Get an Excel spreadsheet as a nested array
[4]  A
[5]  A
[6]  A   ΠIO←1
[7]  A
[8]  A Launch Excel
[9]  A Because 'Excel.Application' is not an APLX class, it looks for a server
[10] A XL←'□' □NEW 'Excel.Application'
[11] A
[12] A In this example we don't want to see Excel. It runs in the background
[13] A XL.visible←0
[14] A
[15] A Open specified spreadsheet
[16] A R←XL.Workbooks.Open SHEETNAME
[17] A
[18] A Get number of rows and columns
[19] A ROWS←XL.ActiveSheet.UsedRange.Rows.Count
[20] A COLS←XL.ActiveSheet.UsedRange.Columns.Count
[21] A
[22] A Get column names. They are called, A B C..Z AA AB AC..AZ BA BB, etc
[23] A COLNAMES←GetColumnNames COLS
[24] A
[25] A Read all data into APL nested array
[26] A R←XL.name □WI 'Range().Value' ('A1:',(≡COLNAMES[COLS]),⌈ROWS)
[27] A
[28] A Ask Excel to quit (reference localized, so reclaim memory at end)
[29] A XL.Quit
    ▽

```

Line 26 of the above function is an example of reading an Array Property, where the parentheses are used to tell APLX that the parameter (in this case a string containing the cell range) should be used to select the particular property required.

Using references to returned objects

The APLX OCX interface allows you to use numeric 'LPDISPATCH' handles which may be returned by some OCX/ActiveX controls or OLE Servers. This usually applies in the case where you create a new object in the external control, and need some way to refer to it. Windows returns a 'handle', which in APL is represented as an integer. You can use the text representation of the integer as part of a hierarchical property or method name to access the newly-created object.

For example, you can create a new Word document using the Add method of the Word application. This returns a handle. In APLX, you can use this as follows:

```
W←'□' □NEW 'Word.Application'  
MYDOC←'W' □WI 'Documents.Add'  
MYDOC  
1797860  
'W' □WI MYDOC, '.Words.Count'  
1
```

Array Properties and Constants

Array Properties

Some OCX controls and OLE servers use special types of properties known as 'Array Properties'. These are rather like indexed arrays in APL, in that one or more parameters are used to determine which one of a set of properties you are reading from or writing to. The parameters are often simply integers, but can be strings or other data types.

To use 'Array Properties', you need to use `⌈WI` rather than dot notation. You pass the parameter(s) as part of the `⌈WI` argument, and put parentheses into the property name string to indicate this. Each pair of parentheses corresponds to one element from the `⌈WI` argument list. An example is shown in line 26 of the `GetSpreadsheet` function above:

```
R←XL.name ⌈WI 'Range().Value' ('A1:',(=COLNAMES[COLS]),⌈ROWS)
```

`Range` is an Array Property, and the parentheses tell APLX to use one element from the nested argument list (in this case a character vector) as the parameter.

Constants

Many OCX controls and OLE servers define named constant values or 'enums' as valid arguments for properties and methods. Often the documentation will refer to the names of these constants without giving the values, which you need to use them from APL.

In APLX, there are two ways you can find out these values. The Control Browser window shows constant values by category for controls and servers which publish them. You can thus look up the value in this dialog.

Alternatively, you can determine the value at runtime using the `Valueof` method. This takes a character vector argument which is the name, and returns the value. For example, Formula One defines a set of enumerated values for file formats. To select the Formula One Version 3 format, you need the constant `F1FileFormulaOne3`. You can either look this up in the documentation, or use the Control Browser to find it, or determine it from within APLX as follows:

```
W.F1.Valueof 'F1FileFormulaOne3'
```

5

Section 9. APLX Multi-tasking

Introduction to APLX multi-tasking

APLX includes built-in support for multi-tasking. This means that a single instance of the APLX program can simultaneously run multiple APL tasks, each with its own workspace and optionally its own session window. (Technically, APLX runs as a single *process*, with multiple *threads*. One thread controls the user-interface, and each APL task has its own thread.)

APLX tasks are of two types, 'top-level' or 'parent' tasks, and 'child' tasks. When APLX starts up, it creates the first top-level task automatically. Further top-level tasks are created using the File menu on the APLX session window, and each has its own session window in which you can enter APL expressions. A top-level task keeps running until one of the following occurs:

- You type)OFF in the task's session window, or your program executes)OFF
- You select 'Close Session' from the File menu
- You close the session window

When a top-level task ends, other top-level tasks are not affected. When all top-level tasks have finished, the APLX program terminates. If you select 'Exit APLX' from the File menu, all tasks terminate.

Child tasks are created under program control using `ⓘWI`. They can either be *background* tasks (with no session window), or alternatively they can be ordinary tasks with their own session windows, although these might be hidden. Child tasks may be terminated in the same ways as top-level tasks, but in addition they can be terminated under program control by the parent task. They will also terminate automatically if their parent task terminates. Child tasks can themselves create further child tasks.

Each task has its own separate memory allocation for the workspace, of a size which you can specify. The tasks execute independently, but you can use *signals* to allow parent and child tasks to communicate with one another. In addition, variables can be shared between tasks. The niladic system function `ⓘUL` returns the number of APLX tasks which are currently running.

Special considerations for Client-Server versions of APLX

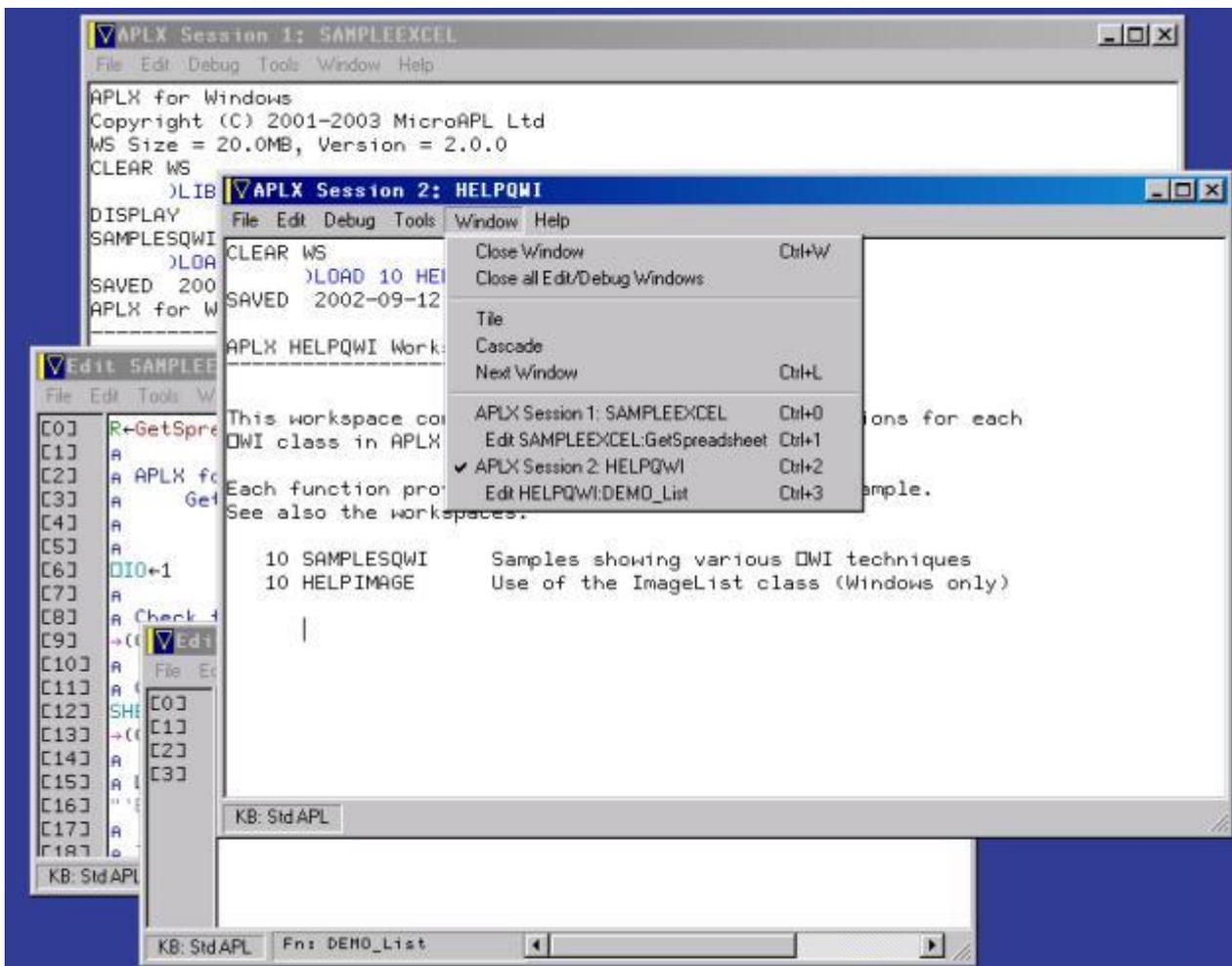
If you are running a Client-Server implementation of APLX, it is possible for your various APLX tasks to run on different machines. This is true both for tasks created using the File menu, and for Child tasks. In the former case, you can specify where the task should be created using the dialog which appears when you select the New Session menu item. In the latter case, you select the Server on which the task should run using the `host` property of the APL Child Task object.

If you are running APLX64, your APL sessions can comprise a mixture of 32-bit and 64-bit tasks.

Creating top-level tasks using the File menu

To create a new top-level task, just select 'New Session' from the File menu. A new session window will open, and you now have two APL tasks running. The window titles indicate which session is which.

You can execute APL expressions and)LOAD workspaces separately in each session window. You can also use the other APLX features such as Edit windows and the Workspace Explorer simultaneously in multiple tasks. To help you keep track of which window belongs to which task, the window title includes the workspace name. In addition, the Window menu shows the window titles associated with each session grouped together and indented. For example, this is how the screen might look if you have two top-level tasks running, and you are editing one function in each task:



Choosing the Host on Client-Server versions of APLX

If you are running a Client-Server implementation of APLX, you can choose on which Server your new APL tasks should run, by going to the APL tab of the APLX Preferences dialog. You can set the default to be a 32-bit APL on the current machine, a 64-bit APL on the current machine (if you have purchased APLX64), or alternatively specify a Server host name which APLX should contact over the network to create the task.

You can also specify that you want to be prompted each time a new session is created, in which case a dialog will appear when you select New Session from the File menu.

Setting the workspace size

When you create top-level task using the File menu, it is allocated the default workspace size which you have set using the Preferences dialog. You can change the workspace size using the `)CLEAR` system command, which optionally takes a parameter which is the desired workspace size in bytes, kilobytes (K), megabytes (M) or gigabytes (G):

```
)CLEAR 800K
WS Size = 800KB
CLEAR WS
)CLEAR 2M
WS Size = 2.0MB
CLEAR WS
```

The valid range is 50KB to 2GB (for 32-bit versions of APLX). For APLX64, the maximum workspace size is limited only by the maximum virtual memory size allowed by the operating system (currently 16384 GB for Windows XP 64). Depending on the operating system and its configuration, you are likely to be limited in the maximum size of workspace which you can allocate.

Creating APL child tasks under program control

Creating a new APL task object

You can create child tasks under program control using the 'APL' System Class. This works as follows:

```
ChildTask←'□' □NEW 'APL'
```

This creates a task object, but the APL task itself does not start running until you call the `Open` method. Before doing that, you can set various properties of the new task:

Specifying whether the task runs in the background

By default, child tasks you create will have their own session window. However, if you set the `background` property to 1, the task will not have a session window and all its output will be thrown away (you must set this property before calling the `Open` method).

Specifying the workspace size

By default, a new child task is allocated the amount of workspace which you have set as the default using the Preferences dialog. The `wssize` property allows you to specify a different size, as a value in bytes. If you ask for a very small value, a minimum value of around 100KB will be allocated. If you ask for a very big value, the operating system may allocate a smaller value. A value of 0 means 'use the default'. You must set the `wssize` property before calling the `Open` method, but you can read it back at any time.

Specify which machine the task should run on

If you are running a Client-Server implementation of APLX, you can set the `host` property of the APL object to specify on which machine the new APL task should run. You must do this before starting the task.

If you have purchased APLX64, your child APL sessions can comprise a mixture of 32-bit and 64-bit tasks. A 32-bit task can start a 64-bit task, and vice versa.

Specifying whether the task's session window is visible

A task which is not a background task will have a session window associated with it, but it may not be visible. You can use the `visible` property (or the `Show` and `Hide` methods) to control whether it is shown, just as you would with any other window. This can be done before or after the `Open` method has been called. You can also use the `where` and `size` properties to set or read the position and size of the task's session window.

Starting the APL task

The actual APL task starts when you call the `Open` method. If the task has a session window, it will be created at this time, and it will appear unless the `visible` property is 0. If the task has a `background` property of 1, nothing will appear. The child task inherits the `ΩMOUNT` table of its parent, so will initially have the same mapping from library numbers to operating-system directories.

Example

This sequence creates a child task with its own visible session window, and a workspace size of approximately 1MB:

```
ChildTask←'Ω' ΩNEW 'APL'  
ChildTask.wssize←1E6  
ChildTask.Open
```

You can enquire about the visibility and coordinates of the session window as follows:

```
ChildTask.visible  
1  
ChildTask.where  
12 32 24 64
```

You can change the size and position of the task's session window as you would with an ordinary window:

```
ChildTask.where←4 10 16 50
```

Using the child task

If the child task has a visible session window associated with it, you can enter commands and APL expressions in it in the normal way. Alternatively, the parent APL task can use it as follows:

Checking the status of the child task:

The `status` read-only property indicates the execution state of the task. It returns one of the following codes:

- 1 The task is not running (i.e. it has not been opened, or has terminated)
- 0 The task is busy executing an APL expression or command
- 1 The task is in desk-calculator mode, awaiting input
- 2 The task is awaiting \square input
- 3 The task is awaiting \square input
- 4 The task is awaiting \square CC input

Causing the child task to execute an expression or command

The `Execute` method allows the parent to cause the child task to execute an APL expression or system command. It takes as an argument a character string which the command to be executed (or passed as input to \square or \square).

The child task should be awaiting input (you can tell whether this is the case by reading the `status` property, or by using the `onReady` callback). If it is not, the command will be placed in a buffer and will be executed when the child task next asks for input. However, there is no queue of commands held; any existing command already in the buffer will be over-written if the child task has not yet executed it. The `Execute` method returns immediately; it does not wait for the command to complete.

In this example, the child task is initially ready for input. It then executes the `)LOAD` command, and is busy for a short time (`status` property is 0). The `status` property then changes back to 1 when it is ready for further input:

```

ChildTask.status
1
ChildTask.Execute ')LOAD 10 SAMPLEEXCEL'
ChildTask.status
0
ChildTask.status
1
```

Note that in a real multi-tasking APLX application you would typically use 'signal' events to allow the parent and child tasks to communicate with each other and synchronize their operations - this is described below. The `Execute` method would typically be used only when starting up the child task.

Reading back the child task's session window contents

The `text` property of the APL object contains the text in the task's session window (assuming it is not a background task). For example, after executing the `)LOAD` above you might have:

```

CONTENTS←ChildTask.text

CONTENTS
CLEAR WS
)LOAD 10 SAMPLEEXCEL
SAVED 2002-09-12 16.43.21
APLX for Windows and Excel Sample - 18th July 2002
-----
This workspace contains some functions to demonstrate how to use APLX to
access Excel spreadsheets via the Microsoft Excel OLE server.

```

... etc

It is a text vector, usually with embedded carriage return (`␣`) characters. The `text` property returns the logical contents of the session window, even if the session window is not visible (however it is always an empty vector for a background task, which does not have a session window). You can also write to this property to change the text in the session window of a child task.

Uniquely identifying a child task

Every task has a unique identifier. You can read the task identifier of a child task using the read-only `taskid` property. It is a scalar integer, and can be used in event handling to identify the task:

```

ChildTask.taskid
53429272

```

The `taskid` property will be zero if the task is not running.

The child task can read its own task identifier using the `taskid` property of its `System` object.

Interrupting a child task under program control

If a child task is executing (`status` property is 0), the `Interrupt` method can be called to interrupt it. It is equivalent to a keyboard interrupt (`break`).

Terminating a child task under program control

A child task may terminate for various reasons. For example, you might use the `Execute` method to cause it to execute an `)OFF` command, or it may run an APL function which ends by executing an `)OFF`. If it has a session window, you can end the task in the same ways as you would a top-level task, using the File menu or by closing the session window.

Alternatively, a parent task can terminate the child by using the `Close` method:

`ChildTask.Close`

This closes any windows belonging to the task, releases the workspace memory, and ends the task. The `status` property reverts to `-1`. If you wish you can then use the `Open` method to start it afresh, or the `Delete` method to destroy the task object altogether (or erase the last reference to it in the workspace).

If a parent task terminates, all its child tasks are automatically terminated as well.

Communication between child and parent tasks

Once a child task is running, events are used to communicate between the child and its parent. These events can arise automatically, or more usually by explicit program action. They are used to trigger the execution of a callback function in the receiving task. (Note: As with all callbacks in APLX, the actual callback is run in a `⎕WE` (wait for events) statement).

The automatic events occur when the state of the child task changes. They are as follows (in each case, `⎕EV[6]` in index origin 1 will contain the unique task ID, as returned by the `taskId` property):

- When the state of the child task changes from executing to awaiting input, an `onReady` event is triggered in the parent's task object. You can associate a callback with this, for example to pass the next command to the child task:

- ```
ChildTask.onReady←'NEXTCMD'
```

This would cause the `NEXTCMD` function to be run when the child task is ready for the next command to be sent to it using the `Execute` method.

- When an untrapped error occurs during function execution (but not desk-calculator mode) in the child task, an `onError` event is triggered in the parent's task object. You can associate a callback with this, for example to write the error to a diagnostic log:

- ```
ChildTask.onError←'ERROR_HANDLER'
```

This would cause the `ERROR_HANDLER` function to be run when an error is encountered in the child task. The error message (in the same format as `⎕ERM`) will be available in `⎕WARG` during the callback. (If you have an `onReady` callback defined as well, this will be also be triggered, after `onError`).

- When the child task is about to execute an expression or command (either typed by the user, or under program control) an `onExecute` event is triggered in the parent's task object. During the callback, `⎕WARG` contains the command or expression which is about to be executed, as a character vector. This can be used for example to write an APL tutorial application when the parent task can see what the user is entering in the child-task session window.
- Finally, when the child task is about to terminate for any reason, an `onClose` event is triggered in the parent's task object.

Signal events

The main method of communication between child and parent tasks is through explicit 'signal' events. This mechanism allows the child task to send a message to the parent, and vice versa. This works as follows:

- If the parent wants to send a signal to the child, it invokes the `Signal` method in the child task object. This causes an `onSignal` event to trigger in the `System` object of the child task.
- Conversely, if the child task wishes to send a signal to the parent, it invokes the `Signal` method in its own `System` object. This causes an `onSignal` event to trigger in the child task object of the parent.

In both cases, the `Signal` method optionally takes an argument, which is any APL array (or an APLX overlay created using `⊞OV`). This is typically used to send a command to the other task, or to return a result to it. Any argument to the `Signal` method is available to the receiving task as the `⊞WARG` system variable during the execution of the callback.

For example, suppose a child task is being used to carry out a time-consuming calculation. It starts by awaiting a signal to indicate that it should do the calculation, with the argument to the signal being the data to work on. When it has completed the calculation, it sends a signal back to the parent with the answer. The following sequence indicates how this might be done.

First we need to do some setup. The parent task attaches a callback to the child task object:

```

▽CALCDONE
[1] 'The answer is ' ⊞WARG
▽

ChildTask.onSignal←'CALCDONE'

```

This will cause the `CALCDONE` function to be run when the child task signals that it has a result available, and the result will thus be printed.

Similarly, the child task attaches a callback to its `System` object, waiting for a signal to indicate that it should carry out the calculation:

```

▽DOCALC;RESULT;DATA
[1] DATA←⊞WARG
[2] RESULT←RUNMODEL DATA
[3] '#' ⊞WI 'Signal' RESULT
▽

'#' ⊞WI 'onSignal' 'DOCALC'
⊞WE ¯1

```

This will cause the `DOCALC` function to be run when the parent signals the child, using the data passed as the argument to the `Signal` method. When the calculation is complete, the child signals back to the

parent. The child then sits in the `QWE` event loop waiting for signals to occur (this takes no CPU resource).

To carry out the calculation, the parent signals the child, and then processes events (if it is not already running under `QWE`):

```
ChildTask.Signal THEDATA
QWE ^1
```

This triggers the child task to run the `DOCALC` function, and on completion the parent's `CALCDONE` function runs:

```
The answer is 42
```

In a real example, the parent task would be responding to other events (for example, user-interface events), and there might be a number of child tasks each simultaneously working on a different run of the model. If you have multiple child tasks running, you can use the task ID (`QEV[6]` in index origin 1) to distinguish between them in the callback function.

There might also be a queue of signals waiting to be processed on either side. Note that there is a limit to the number of events which can be queued, typically around 200 events. If this maximum is exceeded, the oldest events are thrown away.

Using 'delta' properties to share data between tasks

Any property name which starts with the delta character Δ can be used to store your own data in an object. Subject to available memory, you can have as many such properties as you like, and you can store any simple or nested array, or overlay, in each property.

This feature works in a special way for child task objects. A delta property for the System object of a child task is the same as the delta property of the same name in the parent task's child task object. Thus the parent task can assign arbitrary data to a delta property, and the child task can access it through its own System object. If the child task writes to a delta property of its System object, the parent can see the data which has been assigned in the child task object.

For example:

```
Task1←'□' □NEW 'APL'
Task1.wssize←200000 ♦ Task1.background←1
Task1.ΔMESSAGE←'Startup'
Task1.Open
Task1.Execute "'□' □WI 'ΔMESSAGE' 'OK'"
Task1.ΔMESSAGE
```

OK

This technique can be used in a number of ways. When starting a task, it can be used to pass an overlay containing all the functions and variables which the child task will need to run. When the task is running, delta properties can be used to share state information, parameters and results between the parent and the child.

The delta property persists for as long as the child task object exists in the parent task. This means that a child task can write to a delta property, and then terminate. The data will still be available to the parent.

Special considerations for Client-Server versions of APLX

If you are running a Client-Server implementation of APLX, your APL tasks may not all be running on the same machine. However, you can still use delta properties to share data between them. The actual data is held (in memory) on the Client machine, and will be sent over the network to/from the APL task as required. All of this is transparent - you do not need to do anything special for this to work.

The only issue to be aware of is that delta variables (like other \square WI properties) are always held in 32-bit format. Data written from a 64-bit task will be converted to 32-bit form. This means that delta variables cannot be bigger than 2GB, and must not have more than 2147483647 elements. If an integer array contains numbers whose magnitude is bigger than 2^{31} , the array will be converted to floating-point form. Precision may be lost for integers whose magnitude is greater than 2^{53} .

Sharing variables between tasks

As well as passing data using the `signal` method and delta properties, you can also share variables between all the APL tasks you have running (this includes all top-level tasks and all child tasks). You do this using Auxiliary Processor 800, which is built into APLX. All that is required is to share a variable with this processor:

```
      800 □SV0 'TITLE'
2
```

This causes the variable `TITLE` to be held outside the workspace, in a common memory area accessible to any other tasks which use `□SV0` to share the same variable. When you assign to a shared variable of this type, APLX allocates memory for it outside the workspace. If this memory cannot be allocated, an `I0 ERROR` is reported to the APL task. Other than available memory, there are no limits on the size or number of such variables.

The value of the variable will be the last value written to it by any task. This can be any APL array, or an overlay created using `□OV` which contains multiple functions and variables. It persists for as long as any task has the variable shared, and is automatically deleted when it is no longer required (or when APLX exits).

For example:

Share a variable with all other tasks:

```
      800 □SV0 'TITLE'
2
```

Assign some data to it:

```
      TITLE←"Lady Windermere's Fan"
```

Read it back:

```
      TITLE
Lady Windermere's Fan
```

Read it back again:

```
      TITLE
The Importance of Being Earnest
```

Another task has modified it!

Special considerations for Client-Server implementations of APLX

If you are running a Client-Server implementation of APLX, your APL tasks may not all be running on the same machine. In contrast to delta properties, variables shared using Auxiliary Processor 800 are held on the Server machine, so you cannot use this mechanism to share data across the network. In addition, there are separate common data areas for 32-bit and 64-bit tasks.