dyalog APL/W

Dyalog *APL*™ for Windows

Object Reference

Dyadic Systems Ltd

Riverside View
Basing Road, Old Basing
Basingstoke
Hampshire, RG24 7AL
United Kingdom

tel: +44 (0)1256 811125
fax: +44 (0)1256 811130
email: support@dyadic.com
http://www.dyadic.com

Contents

CHAPTER 1

Summary

Graphical User Interface Summary

This chapter provides a summary listing all the objects, properties, events and methods with a brief description.

Table of Objects

Object	Description
ActiveXContainer	Represents the application that is hosting an ActiveXControl
ActiveXControl	Allows you to package a Dyalog APL application as an ActiveX control.
Animation	Plays simple animations from AVI files and resources
Bitmap	A bitmap that can be used to fill an area, to define the appearance of a Button, Menu or MenuItem, or as a background pattern
BrowseBox	Allows the user to browse for and select a folder or other resource
Button	A pushbutton, radio button or checkbox used to perform a task or select an option
Calendar	Provides an interface to the Month Calendar Control.
Circle	Draws circles, arcs and pies
Clipboard	Provides access to the Windows clipboard
ColorButton	Allows the user to select a colour
Combo	Combines a text entry field with a list of available choices for the user to select
ComboEx	An extended version of the Combo object that provides additional features including item images
CoolBand	Represents a band in a CoolBar.
CoolBar	Acts as a container for CoolBand objects.
Cursor	Creates a user-defined cursor that can be associated with an object

Object	Description
DateTimePicker	An editable date/time field with an optional drop-down Calendar
Edit	A single or multi-line edit box for entering, editing, or browsing data
Ellipse	Draws ellipses, elliptical arcs and pies
FileBox	A standard File Selection dialog box
Font	A font resource
Form	A window or dialog box that acts as a container for other objects
Grid	A spreadsheet object for displaying and editing a data matrix
Group	A frame with border and optional title used to group other objects together
Icon	An icon that can be displayed or used when a Form is minimised
Image	A graphical object for displaying bitmaps and icons
ImageList	Represents an array of bitmapped images
Label	Fixed text that the user cannot change
List	Displays a list of items (with or without scrollbar) from which the user can choose
ListView	Displays a collection of items
Locator	Displays a moving/rubberbanding line, rectangle or ellipse for graphics input

Objects (continued)

Object	Description
Marker	Draws markers at a series of points
MDIClient	Provides Multiple Document Interface (MDI) behaviour
Menu	Container object for MenuItems.
MenuBar	Displays a list of pulldown menus.
MenuItem	A component of a Menu that actually performs an action or makes a choice
Metafile	Provides access to Windows Metafiles
MsgBox	Displays a message in a dialog box and waits for the user to respond
NetClient	Represents an instance of a Microsoft .Net Class.
NetType	Used to export a namespace as a Microsoft .Net Class.
OCXClass	Represents an OLE Control
OLEClient	Provides access to an OLE Automation Server
OLEServer	Used to establish a namespace as an OLE Server object that can be used by an OLE Automation client
Poly	Draws lines, polygons and filled areas
Printer	Controls output to a printer
ProgressBar	Used to indicate the progress of a lengthy operation

Objects (continued)

Object	Description
PropertyPage	Represents a single page within a PropertySheet
PropertySheet	Displays a set of PropertyPages
Rect	Draws filled and unfilled rectangles
RichEdit	An edit object with word-processing capabilities
Root	The system object that is the ultimate parent of all others
Scroll	A horizontal or vertical scrollbar
Separator	A horizontal or vertical line in a Menu, or specifies a vertical break in a MenuBar
SM	Allows character-mode applications using $\square SM$ and $\square SR$ to be integrated into Forms and used in conjunction with GUI objects
Spinner	A data entry field that allows a value to be keyed in and updated using spin buttons
Splitter	Divides a container into resizable panes
Static	A frame or box used to contain graphics
StatusBar	Manages a set of StatusField objects
StatusField	Displays context-sensitive help, application and keyboard status
SubForm	A child Form that is constrained within its parent

Objects (continued)

Object	Description
SysTrayItem	Represents an item that you can create in the Windows System Tray
TabBar	Manages a set of TabBtn objects
TabBtn	Tabs (brings forward) an associated SubForm
TabButton	Represents an individual tab or button in a TabControl
TabControl	Provides access to the standard Windows 95/NT tab control
TCPSocket	Provides an interface to TCP/IP
Text	Displays or prints arbitrary text
Timer	Generates events at regular intervals
TipField	Displays pop-up context-sensitive help
ToolBar	Manages a block of controls including Buttons
ToolButton	Represents a button in a ToolControl
ToolControl	Provides a Windows 95/NT ToolBar
TrackBar	Used to display or update a value using a slider and thumb
TreeView	Displays a hierarchical list of items
UpDown	A pair of spin buttons

Objects (continued)

Table of Properties

Property	Description
Accelerator	Specifies a keystroke that will generate a Select event on the object
AcceptFiles	Specifies whether or not the object accepts drag-drop of file icons from Windows Explorer
Active	Determines whether or not an object is currently capable of generating events
Align	Determines the position of text relative to the symbol in a Button. Also used to attach objects to an edge of a Form or Group
AlignChar	Specifies the character(s) on which columns in a Grid are aligned
AlphaBlend	Specifies the level of translucency for a Form (Windows 2000)
APLVersion	Identifies the version of Dyalog APL in use
ArcMode	Determines how arcs are drawn (Ellipse)
Array	Sets or retrieves the contents of the Clipboard in APL format
Attach	Specifies how an object is reconfigured when its parent is resized
AutoArrange	Specifies whether or not items in a ListView are automatically re-arranged when an item is repositioned
AutoBrowse	Specifies whether or not APL attempts to fix functions and variables in an OLEClient namespace when it is created
AutoConf	Governs how a child object reacts to its parent being resized, and whether or not a parent object propagates resizes to its children

Property	Description
AutoExpand	Specifies whether or not rows and columns are automatically added to a Grid
AutoPlay	Specifies whether or not an AVI is played automatically when loaded by an Animation
BandBorders	Specifies whether or not narrow lines are drawn to separate adjacent bands in a CoolBar
BCol	Specifies background colour
Bits	Defines the pattern for a Bitmap, Cursor, or Icon
Border	Determines whether or not an object has a border
BrowseFor	Specifies the type of resource that is the target of a BrowseBox
BtnPix	Associates Bitmaps with Button, Menu and MenuItem objects
Btns	Determines the buttons shown in a MsgBox
CalendarCols	Specifies the colours used for various elements in the Calendar object
Cancel	Used to associate the Esc key with a particular Button
Caption	Specifies a text label for an object.
CaseSensitive	Specifies whether or not string searches in a ComboEx are case-sensitive

Properties (continued)

Property	Description
CBits	Represents the picture in a Bitmap object
CellFonts	Specifies the fonts to be used by the cells in a Grid
CellHeights	Specifies the heights of the cells in a Grid
CellSelect	Specifies cells that user may select in a Grid
CellSet	Identifies which cells in a Grid have values and which are empty
CellTypes	Specifies the type of the cells in a Grid
CellWidths	Specifies the widths of cells in a Grid
Changed	Identifies whether or not an object has been altered by the user
CharFormat	Specifies character formatting for the text in a RichEdit
CheckBoxes	Specifies whether or not check boxes are displayed alongside items in a ListView or TreeView object
Checked	Determines whether or not a check mark is displayed alongside a MenuItem
ChildEdge	Specifies whether or not a CoolBand leaves space above and below its child window
ChildList	Reports the list of objects that can be created as a child of an object
CircleToday	Specifies whether or not a circle is drawn around the Today date in a Calendar object

Properties (continued)

Property	Description
ClassID	Reports the class identifier (CLSID) of an OLEClient or OLEServer object
ClassName	Specifies the name of the OLE object to which an OLEClient object is to be connected
ClipCells	Specifies whether or not a Grid displays partial cells
CMap	Defines a colour map for a Bitmap or Icon
Collate	Specifies whether or not multiple copies of printer output are collated
ColLineTypes	Specifies appearance of vertical grid lines in a Grid
ColorMode	Specifies whether or not printing is done in colour
ColTitle3D	Specifies whether or not a 3-dimensional effect is applied to the column titles in a ListView
ColTitleAlign	Specifies alignment of column titles in a Grid and ListView
ColTitleBCol	Specifies background colour for column titles in a Grid
ColTitleDepth	Specifies the structure for hierarchical column titles in a Grid
ColTitleFCol	Specifies the colour of the text in the column titles of a Grid
ColTitles	Specifies the column titles for a Grid
ColumnWidth	Specifies the with of columns in a multi-column List
Container	The Object Representation of an ActiveXContainer object

Properties (continued)

Property	Description
Coord	Specifies the co-ordinate system for an object
Copies	Specifies the number of copies to be printed
CurCell	Identifies the current cell in a Grid
CurrentColor	Specifies the currently selected colour in a ColorButton
CurrentState	Reports the current state of a TCPSocket object
CursorObj	Associates a Cursor with an object
CustomColors	Identifies the custom colours associated with a ColorButton
CustomFormat	Specifies a custom format for the date/time display in a DateTimePicker
Data	Associates arbitrary data with an object
DateTime	Specifies the value of date/time in a DateTimePicker
DblClickToggle	Specifies whether or not the user must single-click or double-click to toggle the state of a child CoolBand
Decimals	Specifies the number of decimal places for a Numeric field
Default	Nominates a Button to be the default one that is selected when the user presses the Enter key
DefaultColors	Specifies the colours displayed in the colour selection drop-down of a ColorButton
Depth	Specifies the structure of items in a TreeView

Properties (continued)

Property	Description
DevCaps	Reports the device capabilities of the screen or printer
Directory	Specifies the directory for a FileBox
Divider	Controls the presence or absence of a recessed line in a ToolControl object
Dockable	Specifies whether or not an object may be docked and undocked
DockChildren	Specifies a list of objects that may be docked into an object
Docked	Specifies whether or not an object is currently docked in another
DockShowCaption	Specifies whether or not a Form has a title bar when docked as a SubForm
Dragable	Specifies whether or not the user may drag an object with the mouse
DragItems	Specifies whether or not the items in a ListView may be drag-dropped
DrawMode	Provides direct control over the low-level drawing operation performed by graphical objects
Duplex	Specifies whether pages are printed on separate sheets or back-to-back
EdgeStyle	Specifies 3-dimensional appearance
EditImage	Specifies whether or not the edit control portion of the ComboEx displays an image for selected items
EditImageIndent	Specifies whether or not the indents associated with items in a ComboEx object are honoured in the edit control portion of the ComboEx

Properties (continued)

Property	Description
EditLabels	Specifies whether or not the user may edit the labels in a ListView or TreeView
End	Specifies arc ending angles for Circle and Ellipse objects
EnterReadOnlyCells	Specifies whether or not the user may visit read-only cells
Event	Associates an event with a callback function or $\square DQ$ termination
EventList	Reports the names of the events generated by an object.
ExportedFns	Specifies the functions to be exposed as methods by an OLEServer object.
ExportedVars	Specifies the variables to be exposed as properties by an OLEServer object
FCol	Specifies foreground colour
FieldType	Specifies formatting and validation for Edit and Label objects
File	Specifies a filename
FileMode	Specifies the mode (read or write) for a FileBox object
FillCol	Specifies fill colour
Filters	Specifies file filters for a FileBox

Properties (continued)

Property	Description
FirstDay	Specifies the day that is considered to be the first day of the week for a Calendar object
Fixed	Specifies whether a font is fixed-width or proportional
FixedOrder	Specifies whether or not the CoolBar displays CoolBands in the same order
FlatSeparators	Specifies whether or not separators are drawn between buttons in a TabControl object
FontObj	Specifies the font to be used
FontList	Provides a list of available fonts
Formats	Reports the data formats currently available from the Clipboard
FormatString	Specifies a $\square FMT$ specification to format a numeric field
FStyle	Specifies fill style
FullRowSelect	Specifies whether or not the entire row is highlighted when an item in a ListView or a TreeView is selected
GridBCol	Specifies the colour for the unused portion of a Grid
GridFCol	Specifies the colour of (all) the gridlines in a Grid
GridLineFCol	Specifies the colours of the gridlines in a Grid
GridLineWidth	Specifies the widths of the gridlines in a Grid

Properties (continued)

Property	Description
GridLines	Specifies whether or not lines are displayed between items in a ListView object
GripperMode	Specifies whether or not the CoolBand has a gripper bar
HAlign	Specifies horizontal text alignment
Handle	Reports the window handle of an object
HasApply	Specifies whether or not a PropertySheet has an Apply button
HasButtons	Specifies whether or not buttons are shown in a TreeView
HasCheckBox	Specifies whether or not a checkbox is displayed alongside the value in a DateTimePicker
HasEdit	Specifies whether or not a BrowseBox has an edit field.
HasHelp	Specifies whether or not a PropertySheet or PropertyPage has a Help button
HasLines	Specifies whether or not tree lines are drawn in a TreeView
HasTicks	Specifies whether or not ticks are drawn in a TrackBar
HasToday	Specifies whether or not the Today date is displayed in the bottom left corner of a Calendar object
Header	Specifies whether or not a ListView displays column titles

Properties (continued)

Property	Description
HelpButton	Specifies whether or not a Question (?) button appears in the title bar of a Form or SubForm
HelpFile	Reports the name of the help file associated with an OLE Control
Hint	Specifies the text for a context sensitive help message
HintObj	Specifies the object in which to display a Hint
HotSpot	Specifies the hotspot for a Cursor
HotTrack	Specifies whether or not the tabs or buttons in a TabControl object are automatically highlighted by the mouse pointer
HScroll	Determines whether or not an object has a horizontal scrollbar
IconObj	Associates an Icon with a Form to be displayed when the Form is minimised
ImageCount	Reports the number of images in an ImageList
ImageIndex	Maps images in an ImageList to an object or to items in a ListView or TreeView
ImageListObj	Specifies the names of ImageList objects associated with an object
Indents	Specifies the amount by which items in a ComboEx object are indented

Properties (continued)

Property	Description
Index	Specifies the position of items in a Combo or List object, the selected filter in a FileBox, and the sequential position of a CoolBand
Input	Specifies the names of the Edit or Label objects associated with the cells of a Grid
InputMode	Determines the behaviour of cursor movement keys in a Grid
InputModeKey	Specifies the keystroke used to switch input modes in a Grid
InputProperties	Specifies the names of properties of an OCXClass (ActiveX Control) or .NET Class that are to be mapped to the Values property in a Grid
InstanceMode	Specifies how APL attempts to connect an OLEClient to an OLE Server
Interval	Specifies the frequency with which a Timer generates events
Italic	Specifies whether or not a font is italic
Items	Specifies a list of selectable items in a Combo or List object
Justify	Determines how text is justified within an object
KeepBits	Determines how Bitmap, Icon and Cursor objects are stored in the workspace
KeepOnClose	Specifies whether or not objects retain namespace components
LastError	Provides information about the most recent error reported by OLE
LicenseKey	Specifies the license key for an ActiveX control

Properties (continued)

Property	Description
Limits	Specifies the minimum and maximum values for an object
LocalAddr	Specifies the IP address of your computer
LocalAddrName	Specifies the host name of your computer
Locale	Specifies the language in which the OLE server, attached to an OLEClient, exposes its methods and properties
LocalPort	Identifies the port number associated with a TCPSocket object
LocalPortName	Specifies the port name of the local service that you wish to offer as a server
LStyle	Specifies line style
LWidth	Specifies line width
MapCols	Specifies whether button colours in bitmaps and icons in an ImageList are re-mapped to reflect the users colour preferences
Mask	Specifies the mask for a Cursor or Icon
MaskCol	Specifies the transparent colour for a Bitmap or Form
Masked	Specifies whether an ImageList contains masked images
MaxButton	Determines whether or not a Form has a "maximise" button in its title bar

Properties (continued)

Property	Description
MaxDate	Specifies the largest date that may be displayed by a Calendar object
MaxLength	Specifies the maximum number of characters that the user may type into a single-line Edit object
MaxSelCount	Specifies the maximum number of contiguous days that the user may select in a Calendar object
MDIActive	Specifies the active SubForm in an MDI application
MDIMenu	Nominates a particular Menu to be the Windows menu in an MDI application
MetafileObj	Accesses clipboard data in Windows Metafile format
MethodList	Reports the names of methods provided by an OLE Control
MinButton	Determines whether or not a Form has a "minimise" button in its title bar
MinDate	Specifies the smallest date that may be displayed by a Calendar object
MonthDelta	Specifies the number of months by which a Calendar object scrolls when the user clicks its scroll buttons
Moveable	Determines whether or not a Form may be moved to another position on the screen by the user
MultiColumn	Specifies whether or not a List object displays multiple columns

Properties (continued)

Property	Description
MultiLine	Determines whether or not the tabs or buttons will be arranged in multiple flights or multiple rows/columns in a TabControl or ToolControl object
MultiSelect	Specifies whether or not the user can select more than one button in a TabControl at the same time
NewLine	Specifies whether or not a CoolBand starts a new row in a CoolBar
OKButton	Pocket APL only. Speciifes whether a Form has an [OK] button or an [X] button in the top right corner of the title bar.
OLEControls	Reports a list of OLE Controls installed on the computer
OLEServers	Reports the names and CLSIDs of all the OLE Automation servers installed on your computer
OnTop	Specifies that a Form is raised to the front even when it does not have the focus
Orientation	Specifies the orientation of the paper for a Printer object
OtherButton	Specifies whether or not the user may access the Windows Colour Selection dialog box from a ColorButton object
OverflowChar	Specifies the character used to fill a Grid cell when it overflows
PageActive	Specifies the name of the current PropertyPage
PageWidth	Specifies the width of the page in a RichEdit
PaperSize	Specifies the size of paper to be used for printing
PaperSizes	Provides the names and dimensions of the various different paper sizes supported by a Printer object

Properties (continued)

Property	Description
PaperSource	Specifies the name of the paper bin to be used as the paper source for printing
PaperSources	Provides the names of the paper bins installed on a Printer object
ParaFormat	Specifies the paragraph formatting for the text in a RichEdit
Password	Specifies the symbol for a password field
PathWordBreak	Specifies whether or not the edit control portion of the ComboEx will use the forward slash (/), back slash (\), and period (.) characters as word delimiters
Picture	Specifies a Bitmap, Icon or Metafile object to be drawn
PName	Specifies the device for a Printer object
Points	Specifies points for graphical objects
Popup	Specifies the name of a (popup) Menu object that is associated with a ToolButton
Posn	Specifies the position of an object within its parent
PrintList	Reports the list of installed printers
PrintRange	Specifies the range of pages to be printed
ProgressStyle	Specifies the appearance of a ProgressBar control
PropList	Reports the list of properties that are applicable to the object

Properties (continued)

Property	Description
QueueEvents	Specifies whether or not incoming events for an instance of an OCXClass object (an ActiveX control) are queued
Radius	Specifies the radius for a Circle
Range	Specifies the range of a scrollbar
ReadOnly	Specifies whether or not the user may modify text in an Edit or Spinner or the state of a Check or Radio Button
RealSize	Specifies the size for a placeable Metafile
RemoteAddr	Specifies the IP address of the remote computer
RemoteAddrName	Specifies the host name of the remote computer to which you wish to make a connection
RemotePort	Identifies the port number associated with a service on a remote computer
RemotePortName	Specifies the port name of the remote service to which you wish to make a connection
ReportInfo	Specifies associated data to be displayed in a ListView
ResizeCols	Specifies whether or not the user may resize Grid columns
ResizeColTitles	Specifies whether or not the user may resize Grid column titles
ResizeRows	Specifies whether or not the user may resize Grid rows
ResizeRowTitles	Specifies whether or not the user may resize Grid row titles
Resolution	Determines the print resolution for a Printer object

Properties (continued)

Property	Description
Resolutions	Reports the available printer resolutions of a Printer object
Rotate	Specifies the angle of rotation for a Font
RowLineTypes	Specifies appearance of horizontal grid lines in a Grid
Rows	Specifies the number of rows displayed in the drop-down list part of a Combo
RowTitleAlign	Specifies the alignment of row titles in a Grid
RowTitleBCol	Specifies background colour for row titles in a Grid
RowTitleDepth	Specifies the structure for hierarchical row titles in a Grid
RowTitleFCol	Specifies the colour of the row title text in a Grid
RowTitles	Specifies the row titles for a Grid
RTFText	Specifies the contents of the clipboard or a RichEdit in Rich Text Format (RTF)
RunMode	Specifies the way in which an OLEServer object serves multiple clients
ScrollOpposite	Specifies that unneeded tabs scroll to the opposite side of a TabControl
SelDate	Identifies the range of dates that is currently selected in a Calendar object
SelImageIndex	Determines which bitmapped images in an ImageList correspond to items in a TreeView object when the item is selected

Properties (continued)

Property	Description
SelItems	Specifies the selected item(s) in a List or Combo
SelRange	Specifies a selection range for a TrackBar
SelText	Specifies the selected text in an Edit or Combo
ServerVersion	Specifies the version number of an OLEServer object
ShowCaptions	Specifies whether or not the captions of individual ToolButton objects are drawn
ShowDropDown	Specifies whether or not a drop-down menu symbol is drawn in ColorButton and ToolButton objects
ShowInput	Specifies how Grid cells are displayed
ShowSession	Specifies whether or not the APL Session window is displayed when an OLEServer object is started by an OLE client
ShowThumb	Specifies whether or not the thumb in a TrackBar is visible
SingleClickExpand	Specifies whether or not an item in a TreeView control is expanded when the user selects the item
SIPMode	Pocket APL only. Specifies the behaviour of the Input Panel.
SIPResize	Pocket APL only. Specifies how a Form resizes when the Input Panel is rasied and lowered.
Size	Specifies the size of an object
Sizeable	Specifies whether or not the user may resize an object using the mouse

Properties (continued)

Property	Description
SocketNumber	An integer whose value is the Window handle of the socket attached to the TCPSocket object
SocketType	Specifies the type of the TCP/IP socket
SortItems	Specifies whether or not the Items in a List object are sorted.
SplitObj1	Specifies the name of an object managed by a Splitter
SplitObj2	Specifies the name of an object managed by a Splitter
Start	Specifies start angles for arcs of Circle and Ellipse objects
StartIn	Specifies the start point and root for a BrowseBox object
State	Specifies the state of a Button or Form
Step	Specifies the increments for movement within an object
Style	Specifies the style of an object
SysMenu	Determines whether or not a Form has a standard system menu in its title bar
TabFocus	Specifies the focus behaviour for the TabControl object
TabJustify	Specifies the positions at which the picture and caption are drawn within a TabButton
TabObj	Specifies the name of the SubForm associated with a TabBtn or TabButton
TabSize	Specifies the size of fixed size tabs or buttons in a TabControl object

Properties (continued)

Property	Description
Target	Specifies the chosen folder or other resource selected by the user in a BrowseBox object
Text	Specifies/reports the text in an Edit, MsgBox, or in the edit field of a Combo
TextSize	Reports the bounding rectangle for text
Thumb	Specifies the position of the thumb in an object
ThumbRect	Reports the position and size of the thumb in a TrackBar
TickAlign	Specifies the position of tick marks in a TrackBar
TickSpacing	Specifies the spacing of tick marks in a TrackBar
Tip	Specifies the text for a pop-up help message
TipObj	Specifies the object in which to display the Tip
TitleHeight	Specifies the height of the column titles in a Grid
TitleWidth	Specifies the width of the row titles in a Grid
Today	Specifies today's date in a Calendar object
ToolboxBitmap	Specifies a bitmap image (tool) for a COM object
TrackRect	Reports the position and size of the slider in a TrackBar

Properties (continued)

Property	Description
Translate	Specifies whether or not character data is translated to and from □*AV*
Transparent	Specifies whether or not a ToolControl is transparent
Type	Specifies the type of an object
TypeLibID	Specifies the value of the globally unique identifier (GUID) of the Type Library associated with a COM object
TypeLibFile	Specifies the name of the file in which the Type Library for a COM object is stored
TypeList	Reports the names of data types associated with an OLE Control
Underline	Specifies whether or not a font is underlined
UndocksToRoot	Specifies the parent adopted by an object when its Type changes to a Form as a result of an undocking operation
UpperCase	Specifies that property names are to be reported in uppercase
ValidIfEmpty	Specifies whether or not an empty numeric object is deemed to be valid
VAlign	Specifies vertical text alignment
Value	The value of a number, date or time in an Edit or Label object
Values	The data matrix in a Grid

Properties (continued)

Property	Description
VariableHeight	Specifies whether or not a CoolBar displays bands at the minimum required height, or all the same height
View	Specifies the appearance of a ListView
Visible	Specifies whether or not an object is currently visible
VScroll	Specifies whether or not an object has a vertical scrollbar
WantsReturn	Determines the behaviour of the Enter key in an Edit or RichEdit
WeekNumbers	Specifies whether or not a Calendar object displays week numbers
Weight	Specifies the weight (boldness) of a font
WordFormat	Specifies the word formatting for text in a RichEdit
Wrap	Determines how an object behaves when its value overflows
XRange	Specifies origin and scale on the x-axis
Yield	Specifies how frequently Dyalog APL/W yields control
YRange	Specifies origin and scale on the y-axis

Properties (continued)

Table of Events

Event		Description
ActivateApp	139	User has switched to or from the APL application
AddCol	153	User has appended a column to a Grid object. Also used to insert a new column under program control
AddRow	152	User has appended a row to a Grid object. Also used to insert a new row under program control.
AmbientChanged	533	Reported when any of the ambient properties change in an application hosting an ActiveXControl object.
AnimStarted	294	Reported by an Animation object just before an AVI clip starts playing
AnimStopped	295	Reported by an Animation object just after an AVI clip has stopped playing
BadValue	180	User has attempted to leave an Edit object containing text that is invalid in relation to its FieldType
BeginEditLabel	300	User has started to edit an item in a ListView or TreeView
CalendarDblClick	273	Reported when the user double-clicks the left mouse button over a Calendar object
CalendarDown	271	Reported when the user depresses the left mouse button over a Calendar object
CalendarMove	274	Reported when the user moves the left mouse button over a Calendar object
CalendarUp	272	Reported when the user releases the left mouse button over a Calendar object

Event		Description
CellChange	150	User is modifying the contents of a cell in a Grid object
CellChanged	164	User has modified the contents of a cell in a Grid object
CellDblClick	163	User has double-clicked the mouse on a cell in a Grid
CellDown	161	User has depressed a mouse button over a cell in a Grid
CellError	157	User has input invalid data into a cell in a Grid
CellMove	151	User has moved to a new cell in a Grid object
CellUp	162	User has released a mouse button over a cell in a Grid
Change	36	User has altered the text in an Edit or Combo
ClickComment	225	Generated when the user clicks the mouse in a Grid comment window
ClipChange	120	Data in the clipboard has changed
Close	33	A Form is about to be closed
CloseUp	46	Reported by a DateTimePicker object just before the drop-down calendar is hidden
ColorChange	430	User has changed the colour in a ColorButton object
ColumnClick	320	User has clicked on a column heading in a ListView
Configure	31	The configuration (position and/or size) of an object is about to change

Events (continued)

Event		Description
ContextMenu	410	Reported when the user performs the standard Windows action to display a context menu
Create	34	Reported immediately after an object has been created
DateTimeChange	267	Reported by a DateTimePicker object when the user changes the DateTime value
DDE	50	A DDE message has been received or sent
DisplayChange	137	User has changed screen resolution and/or number of colours
DockAccept	483	Reported by a host object just before it accepts a client object docking operation
DockCancel	485	Reported by a client object when the user aborts a docking operation by pressing Escape
DockEnd	484	Reported by a client object after it has been successfully docked in a host object
DockMove	481	Reported by a host object when a dockable object (the client) is dragged over it
DockRequest	482	Reported by a client object just before it is docked in a host object, when the user releases the mouse button
DockStart	480	Reported by a dockable object when the user starts to drag it using the mouse

Events (continued)

Event		Description
DragDrop	11	User has moved an object using a drag & drop operation
DropDown	45	Reported when the user clicks the drop-down button in a Combo, ComboEx, DateTimePicker or Menu object, just before the drop-down list, calendar or menu is displayed
DropFiles	450	User has drag-dropped file icons onto the object
DropObjects	455	User has drag-dropped APL object icons onto the object
DyalogCustomMessage1	95	Allows external applications and dynamic link libraries to insert events into the Dyalog APL/W message queue
EndEditLabel	301	User has finished editing an item in a ListView or TreeView
EndSplit	282	Reported when user releases the left mouse button to signify the end of a drag operation on a Splitter object
ExitApp	132	User has selected End Task from the Windows Task List
ExitWindows	131	User has requested Windows to terminate
Expanding	302	Reported by a TreeView when it is about to expand its tree
Expose	32	Part or all of a Form or a Static has been exposed and may need to be redrawn
FileBoxCancel	72	User selected the Cancel button in a FileBox

Event		Description
FileBoxOK	71	User selected the OK button in a FileBox
FontCancel	242	User cancelled a font selection (ChooseFont)
FontOK	241	User executed a font selection (ChooseFont)
FrameContextMenu	411	Reported when the user clicks and releases the right mouse button over the non-client area of an object, e.g. the title bar in a Form
GetDayStates	266	Reported when a Calendar object requires the APL program to provide day state information
GotFocus	40	An object has received the input focus
GridCopy	191	User has copied a block of Grid cells to the clipboard
GridCut	190	User has cut a block of Grid cells to the clipboard
GridDelete	193	User has deleted a block of Grid cells
GridDropSel	195	User has drag-dropped a block of Grid cells
GridKeyPress	24	Reported when the user presses a key in a Grid
GridPaste	192	User has pasted data into a Grid
GridPasteError	194	User has attempted to paste inappropriate data into a Grid
GridSelect	165	User has selected a block of Grid cells

Events (continued)

Event		Description
Help	400	User has requested help on the object
HideComment	224	Generated just before a comment window is hidden as a result of the user moving the mouse-pointer
HScroll	39	User has requested a movement of the thumb in the horizontal scrollbar of a Form
Idle	130	Generated when system is idle
IndexChanged	210	The Index property of a Grid has changed
ItemDblClick	342	User has double-clicked on an item in a ListView or TreeView
ItemDown	340	User has pressed the left mouse button over an item in a ListView or TreeView
ItemUp	341	User has released the left mouse button over an item in a ListView or TreeView
KeyError	23	User has pressed an invalid key in an object
KeyPress	22	User has pressed a key on the keyboard
Locator	80	User terminated interaction with a Locator object
LostFocus	41	Object has lost the input focus
MDIActivate	42	Generate when an MDI SubForm becomes the active one
MDIDeactivate	43	Generated when an MDI SubForm is deactivated

Events (continued)

Event		Description
MouseDblClick	5	User has double-clicked a mouse button (clicked twice in quick succession)
MouseDown	1	User pressed mouse button down
MouseEnter	6	User has moved the mouse into the object
MouseLeave	7	User has moved the mouse out of the object
MouseMove	3	User has moved the mouse
MouseUp	2	User released a mouse button
MouseWheel	8	User rotated the mouse wheel
MsgBtn1	61	User selected first button in a MsgBox
MsgBtn2	62	User selected second button in a MsgBox
MsgBtn3	63	User selected third button in a MsgBox
PageActivate	360	User has switched to this PropertyPage
PageApply	350	User has pressed the Apply button in a PropertySheet
PageBack	353	User has pressed the Back button in a PropertySheet
PageCancel	351	User has pressed the Cancel button in a PropertySheet
PageChanged	356	User has altered the data in a PropertyPage

Events (continued)

Event		Description
PageDeactivate	361	User has switched to another PropertyPage
PageFinish	355	User has pressed the Finish button in a PropertySheet
PageHelp	352	User has pressed the Help button in a PropertySheet
PageNext	354	User has pressed the Next button in a PropertySheet
PreCreate	534	Reported when an instance of an ActiveXControl is created
Protected	470	User has attempted to change protected text in a RichEdit
Retracting	304	A TreeView is about to collapse part of its tree
Scroll	37	User has requested a movement of the thumb in a scrollbar
SelDateChange	265	Reported when the user changes the date that is selected in a Calendar object
Select	30	User has selected the object
SetColSize	176	User has changed the width of a column in a Grid or ListView
SetItemPosition	322	User has drag-dropped an item in a ListView
SetRowSize	175	User has changed the height of a row in a Grid
SetSpinnerText	421	Reported just before the text is changed in a Spinner

Events (continued)

Event		Description
SetWizard	365	User has clicked the Next or Back button in a Wizard PropertySheet
ShowComment	223	User has hovered the mouse pointer over a commented Grid cell
Spin	420	User has incremented a Spinner
Splitting	281	Reported while a Splitter object is being dragged, between a StartSplit and an EndSplit
StartSplit	280	Reported when the user depresses the left mouse button over a Splitter object
StateChange	35	A Form is about to change state
SysColorChange	134	The system colour scheme has changed
TCPAccept	371	Reported when a client connects to a server TCPSocket object
TCPClose	374	Reported when the remote end of a TCP/IP connection breaks the connection
TCPConnect	372	Reported when a server accepts the connection of a client TCPSocket object
TCPError	370	Generated when a fatal TCP/IP error occurs
TCPGotAddr	377	Reported when a name is resolved to an IP address
TCPGotPort	378	Reported when a port name is resolved to a port number
TCPReady	379	Reported when the TCP/IP buffers are free and there is no data waiting to be sent in the internal APL queue

Events (continued)

Event		Description
TCPRecv	373	Reported when data is received by a TCPSocket object
ThumbDrag	440	User has dragged the thumb in a TrackBar
Timer	140	Event generated by a Timer object
VScroll	38	User has requested a movement of the thumb in the vertical scrollbar of a Form
WinIniChange	133	WIN.INI has changed

Events (continued)

Table of Methods

Method		Description
Abort	103	Aborts a print job
AddChildren	310	Adds child items to an item in a TreeView
AddComment	220	Associates a comment with the cell in a Grid
AddItems	308	Adds items to a TreeView
Animate	29	Produces special effects when showing or hiding objects
AnimClose	291	Closes the AVI file that is currently loaded in an Animation object
AnimOpen	290	Opens an AVI file in an Animation object
AnimPlay	292	Plays an AVI clip in an Animation object
AnimStop	293	Stops playing an AVI clip in an Animation object
Browse	585	Causes APL to browse the object's type library and to fix functions and variables in the OLEClient namespace
CancelToClose	367	Changes the buttons in a PropertySheet
CellFromPoint	200	Converts from Grid co-ordinates to cell co-ordinates
ChooseFont	240	Displays a font selection dialog box
ColChange	159	Sets new values for a complete column of cells in a Grid

Method		Description
DateToIDN	264	Converts a date from $\Box TS$ format into an IDN suitable for use in a Calendar object
DelCol	155	Deletes a column from a Grid object
DelComment	221	Removes the comment associated with a cell in a Grid
DeleteChildren	311	Removes child items from a parent item in a TreeView
DeleteItems	309	Removes items from a TreeView
DeleteTypeLib	521	Removes a loaded Type Library from the workspace
DelRow	154	Deletes a row from a Grid object
Detach	270	Detaches the GUI component from an object
DuplicateColumn	178	Duplicates a column in a Grid
DuplicateRow	177	Duplicates a row in a Grid
FileRead	90	Causes a graphical object to be read from a file
FileWrite	91	Causes a graphical object to be written to a file
Flush	135	Forces any objects that have been created to be displayed
GetBuildID	192	Used to obtain the Build ID of a file (e.g. DYALOG.EXE)
GetCellRect	201	Returns the rectangle associated with a Grid cell

Methods (continued)

Method		Description
GetCommandLine	145	Returns the command line that was used to start the current Dyalog APL session or application
GetCommandLineArgs	148	Returns the command line that was used to start the current Dyalog APL session or application and its arguments (as a nested array)
GetComment	222	Retrieves the comment associated with a cell in a Grid
GetEnvironment	510	Obtains APL start-up parameters
GetEventInfo	551	Obtains information about an Event of an OLE Control
GetFocus	511	Returns the name of the object that has the input focus
GetItemHandle	313	Obtains the Windows handle of an item in a TreeView
GetItemPosition	323	Obtains the position of an item in a ListView
GetItemState	306	Obtains the status of an item in a TreeView
GetMethodInfo	552	Obtains information about a method of an OLE Control
GetMinSize	275	Obtains the minimum size that you must specify for a Calendar object for it to display a complete month
GetParentItem	312	Obtains the index of the parent of an item in a TreeView
GetPropertyInfo	550	Obtains information about a property of an OLE Control
GetTextSize	92	Obtains the size of the bounding rectangle of a text item in a given font

Methods (continued)

Method		Description
GetTypeInfo	553	Obtains information about a TypeList of an OLE Control
GetVisibleRange	262	Obtains the range of dates that is currently visible in a Calendar object
GreetBitmap	138	Used to display or remove a bitmap during APL start-up
IDNToDate	263	Used to convert a date from an IDN into $\Box TS$ format
ListTypeLibs	520	Returns a list of loaded Type Libraries
LockColumns	227	Locks columns in a Grid
LockRows	226	Locks rows in a Grid
MakeGIF	261	Generates an uncompressed GIF representation of a picture from a Bitmap object
MakePNG	260	Generates a PNG representation of a picture from a Bitmap object
MDIArrange	112	Causes an MDIClient to arrange the icons associated with its minimised SubForms
MDICascade	110	Causes an MDIClient to rearrange its SubForms as overlapping windows
MDITile	111	Causes an MDIClient to rearrange its SubForms into a row or column
NameFromHandle	136	Obtains the name of an object from its Handle property
NewPage	102	Throws a new page on a Printer
OLEAddEventSink	540	Connects a named event sink to a COM object

Methods (continued)

Method		Description
OLEDeleteEventSink	541	Disconnects a named event sink from a COM object
OLEListEventSinks	542	Returns the names of event sinks that are currently connected to a COM object
OLEQueryInterface	543	Used to obtain the methods and properties associated with a particular interface that is provided by a COM object
OLERegister	530	Used to register an OLEServer object
OLEUnregister	531	Used to unregister an OLEServer object
Print	100	Spools Printer output
ProgressStep	250	Increments the thumb in a ProgressBar
RowChange	158	Sets new values for a complete row of cells in a Grid
RTFPrint	461	Prints the contents of a RichEdit
RTFPrintSetup	460	Invokes the print set-up dialog box for a RichEdit
SetCellSet	171	Sets the value of the CellSet property of a Grid for a particular cell
SetCellType	156	Changes the CellTypes property for a specific cell in a Grid object
SetEventInfo	547	Used to register an event that may be generated by an ActiveXControl object

Methods (continued)

Method		Description
SetFinishText	366	Sets the caption of the Finish button in a Wizard-style PropertySheet
SetFnInfo	545	Used to describe an APL function that is to be exported as a method, or as a property, of an ActiveXControl object
SetItemImage	315	Allocates a picture icon to an item in a TreeView
SetItemState	307	Sets the status of an item in a TreeView
SetMethodInfo	546	Used to describe a method that is exported by a COM object
SetPropertyInfo	554	Used to describe a property that is exported by a COM object
Setup	101	Displays printer Setup dialog box
SetVarInfo	546	Used to describe an APL variable that is to be exported as a property of an ActiveXControl object
ShowHelp	580	Displays a help topic for an OLE Control
ShowItem	316	Displays an item in a TreeView
ShowProperties	560	Displays a property sheet for an OLE Control

Methods (continued)

Method		Description
ShowSIP	25	PockAPL only. Raises or lowers the Input Panel.
TCPGetHostID	376	Used to obtain the IP Address of your PC
TCPSend	375	Used to send data to a remote process connected to a TCPSocket object
TCPSendPicture	380	Transmits a picture represented by a Bitmap object to a TCP/IP socket
Undo	170	Reverses the last change made to a Grid object
Wait	147	Executes $\square DQ$ on an object

Methods (continued)

CHAPTER 2

A-Z Reference

Graphical User Interface A-Z Reference

This chapter provides a complete reference in alphabetical order to the **objects**, **properties**, **events** and **methods** through which Dyalog APL supports the Graphical User Interface.

Abort   Method 103

Applies to   Printer

This method causes the current print job to be aborted and all pending output to be discarded.

The Abort method is niladic.

If you attach a callback function to this event and have it return a value of 0, the print job will continue.

Accelerator   Property

Applies to   ActiveXControl, Bitmap, Button, Calendar, Circle, Clipboard, ColorButton, Combo, ComboEx, Cursor, DateTimePicker, Edit, Ellipse, Form, Grid, Group, Icon, Image, Label, List, ListView, Locator, Marker, MDIClient, Menu, MenuItem, Metafile, Poly, Printer, ProgressBar, Rect, RichEdit, Scroll, Spinner, Static, StatusBar, StatusField, SubForm, TabBar, TabBtn, Text, ToolBar, ToolButton, TrackBar, TreeView, UpDown

This property specifies a keystroke that, when pressed by the user, will generate a Select event on an object. It applies to all objects whether or not they possess a "natural" Select event. You can therefore associate a keystroke with an arbitrary action on any object you desire.

The Accelerator property is a 2-element integer vector. The first element is a key number which is the number by which Windows *knows* the key. The second element is the shift state which is the sum of 1 (Shift key), 2 (Control key) and 4 (Alt key).

For example, to attach the keystroke Ctrl+A to an object, you would set its Accelerator to (65 2). To attach the keystroke Shift+Ctrl+F1 (key number 112), you would set its Accelerator to (112 3). Key numbers may be obtained by inspecting the Dyalog APL/W Input Tables in the APLKEYS directory, or by displaying the messages generated by the KeyPress event.

Note that a keystroke used as an Accelerator will **not** generate a KeyPress event.

AcceptFiles   Property

Applies to   ActiveXControl, Animation, Button, Combo, ComboEx,
                      DateTimePicker, Edit, Form, Grid, Group, Image, Label, List,
                      ListView, ProgressBar, PropertyPage, RichEdit, Scroll, SM, Spinner,
                      Static, StatusBar, SubForm, TabBar, ToolBar, TrackBar, TreeView,
                      UpDown

The AcceptFiles property is Boolean and specifies whether or not an object will accept a
file drag/drop operation. Its default value is 0. If set to 1, the object will report a
DropFiles event when file icons are dropped on it.

ActivateApp   Event 139

Applies to   Root

If enabled, this event is reported when the user switches to or from a Dyalog APL
application.

The event is reported for information only and cannot be modified or annulled by the
result of a callback function.

The event message reported as the result of *□DQ*, or supplied as the right argument to
your callback function, is a 3-element vector as follows :

[1] Object name	:	character vector
[2] Event code	:	'*ActivateApp*' or 131
[3] Activation flag	:	0 or 1

The Activation flag is 0 when the user switches *from* Dyalog APL to another application

The Activation flag is 1 when the user switches *to* Dyalog APL from another
application.

Active   Property

Applies to   ActiveXControl, Animation, Button, Calendar, ColorButton, Combo, ComboEx, DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView, Menu, MenuItem, ProgressBar, PropertyPage, PropertySheet, RichEdit, Scroll, Spinner, Splitter, Static, StatusBar, SubForm, TabBar, TabBtn, Text, Timer, ToolBar, ToolButton, TrackBar, TreeView, UpDown

This property specifies whether or not an object is currently responsive to user actions. It is a single number with the value 0 (object is inactive and does not generate events) or 1 (object is active and capable of generating events). The default is 1.

Setting Active to 0 disables the object (and all its children), even though the object may be referenced in the argument to `⎕DQ`. It is therefore possible to deactivate an object from a callback function.

In general, the text associated with an object whose Active property is 0 is displayed in the appropriate inactive colour.

ActiveXContainer Object

Purpose	The ActiveXContainer object represents the application that is currently hosting an instance of an ActiveXControl object.
Parents	ActiveXControl
Children	(None)
Properties	Type, Event, FontObj, FCol, BCol, Data, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	AmbientChanged, Close, Create
Methods	Detach, OLEQueryInterface

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

An ActiveXContainer object is created using the Container property of the ActiveXControl object. For example, the following expression, executed within an ActiveXControl instance, creates an ActiveXContainer named *'CONT'*

> *'CONT' ⎕NS ⎕WG'Container'*

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

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

ActiveXControl Object

Purpose	The ActiveXControl object represents a Dyalog APL namespace as an ActiveX control.
Parents	Form
Children	ActiveXContainer, Animation, Bitmap, BrowseBox, Button, Calendar, Circle, Clipboard, ColorButton, Combo, ComboEx, CoolBar, Cursor, DateTimePicker, Edit, Ellipse, FileBox, Font, Form, Grid, Group, Icon, Image, ImageList, Label, List, ListView, Locator, Marker, MDIClient, Menu, MenuBar, Metafile, MsgBox, OCXClass, OLEClient, OLEServer, Poly, Printer, ProgressBar, PropertySheet, Rect, RichEdit, Scroll, Spinner, Splitter, Static, StatusBar, SubForm, TabBar, TabControl, TCPSocket, Text, Timer, TipField, ToolBar, ToolControl, TrackBar, TreeView, UpDown
Properties	Type, ClassName, Posn, Size, Coord, Border, Active, Visible, Event, Dragable, FontObj, FCol, BCol, Picture, CursorObj, AutoConf, YRange, XRange, Data, TextSize, EdgeStyle, Handle, Translate, Accelerator, AcceptFiles, ClassID, Container, KeepOnClose, HelpFile, ToolboxBitmap, TypeLibID, TypeLibFile, LastError, Redraw, MethodList, ChildList, EventList, PropList
Events	AmbientChanged, Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, FontCancel, FontOK, GotFocus, Help, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, PreCreate, Select
Methods	Animate, ChooseFont, Detach, GetFocus, GetTextSize, SetEventInfo, SetFnInfo, SetVarInfo, ShowSIP

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

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

To make an ActiveXControl available to another application, you must select *Make OCX* from the Session *File* menu. This creates an .OCX file that contains your entire workspace and all of the ActiveXControls within it. The OCX file is then registered, thereby installing the ActiveXControls on your computer. If an ActiveXControl contains one or more embedded OLEServer objects, these are saved and registered too.

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

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

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

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

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

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

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

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

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

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

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

The ToolboxBitmap property specifies the name of a Bitmap object that may be used by a host application during its design mode. For example, if you add an ActiveX control to the Microsoft Visual Basic development environment, its bitmap is added to the toolbox. The Bitmap should therefore be of an appropriate size, usually 24 x 24 pixels.

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

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

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

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

AddChildren   Method 310

Applies to   TreeView

This method is used to add child items to an item in a TreeView object

The argument to AddChildren is a 3, 4 or 5 element array as follows:

[1] Item number: Integer.

[2] New items: Vector of character vectors.

[3] Depth vector: Integer vector.

[4] Picture vector   Integer vector.

[5] Selected picture vector   Integer vector.

Item number specifies the index of the item to which the child items are to be added.

New items is a vector of character vectors containing the labels for the new child items.

Depth vector is an integer vector specifying the depth of each of the new items relative to the parent item to which they are being added. The first element of this array must be 0.

Picture vector and *Selected picture vector* are optional and specify values of ImageIndex and SelImageIndex respectively for each of the new items.

The result is the index at which the first new item has been inserted.

AddCol Event 153

Applies to Grid

If enabled, this event is reported by the Grid object if the user presses the Cursor Right key, and the current cell (CurCell) is within the last column on the Grid. The default action is to append a new column to the contents of the Grid. If you attach a callback function to this event and have it return a value of 0, a new column will not be appended to the Grid. Note that the event will not be generated unless the second element of the AutoExpand property is set to 1.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	*'AddCol'* or 153
[3] Column number:	number of the new column (integer)

An application may insert a new column into a Grid by calling AddCol as a method. The argument is a 1 to 7-element array as follows:

[1] Column number:	number of the new column (integer)
[2] Column title:	character vector or matrix
[3] Column width:	integer
[4] Undo flag:	0 or 1
[5] Resize flag:	0 or 1
[6] Title colour:	negative scalar integer or 3-element RBG
[7] Line type:	integer

If you are using default column headings, *Column title* will be ignored and the columns will be re-labelled with the default titles. If you have set ColTitles, the title you specify will be inserted. If you omit the *Column title* parameter, a blank title will be inserted.

Similarly, if you have not previously set CellWidths, ResizeCols, ColTitleFCol or ColLineTypes, or if you have given them a scalar value, the corresponding parameter will be ignored. However, if you have specified CellWidths, ResizeCols, ColTitleFCol or ColLineTypes to be a *vector*, the number you specify in the corresponding parameter will be inserted into the appropriate property vector. If you omit to specify *Column width* for the new column, it will be assigned a default value; new values for the other properties default to 0.

Undo flag (default 1) specifies whether or not the addition of the new column may subsequently be undone by an Undo event.

To insert a new column before the first one, you must specify the *Column number* as 1 (or 0 if `⎕IO` is 0). To add a new column after the last one, you may specify any number greater than the current number of columns. The data in the new column will be set to 0 if the Values property is numeric, or to an empty character vector otherwise.

AddComment   Method 220

Applies to   Grid

This method is used to add a new comment.

The argument to AddComment is a 3, 4 or 5 element array as follows:

[1] Row:		integer
[2] Column:		integer
[3] Comment text		character array
[4] Height in pixels		integer
[5] Width in pixels		integer

For example, the following statement associates a comment with the cell at row 2, column 1; the text of the comment is "Hello", and the size of the comment window is 50 pixels (high) by 60 pixels (wide).

```
F.G.AddComment 2 1 'Hello' 50 60
```

The height and width of the comment window, specified by the last 2 elements of the argument are both optional. If the cell already has an associated comment, the new comment replaces it.

You can use a Dynamic Function to add several comments in one statement; for example:

```
(1 2)(2 3){F.G.AddComment α,⊂ω}¨'Hello' 'Goodbye'
```

Note that just before the comment is displayed, the Grid generates a ShowComment event which gives you the opportunity to (temporarily) change the text and/or window size of a comment dynamically.

The comment text specified by the 5th element of the argument to $\square NQ$ must be a simple character scalar, vector, matrix or vector of vectors. Text specified by a simple character vector will be wrapped automatically if necessary. A matrix or vector of vectors may be used to explicitly specify multi-line text. If the array is a vector whose first element is an opening brace ({), the text is assumed to be in rich-text format (RTF) and is displayed accordingly. Note that there is no way for the user to scroll the text in the comment window and it is entirely your responsibility to ensure that the size of the window is appropriate for its contents.

AddItems   Method 308

Applies to   TreeView

This method is used to add items to a TreeView object

The argument to AddItems is a 3,4 or 5-element array as follows:

[1] Item number:	Integer.
[2] New items:	Vector of character vectors.
[3] Depth vector:	Integer vector.
[4] Picture vector	Integer vector.
[5] Selected picture vector	Integer vector.

Item number specifies the index of the item to which the child items are to be added.

New items is a vector of character vectors containing the labels for the new child items.

Depth vector is an integer vector specifying the depth of each of the new items relative to the parent item to which they are being added. The first element of this array must be 0. This element may be omitted. If so, it is assumed to be all 0s.

Picture vector and *Selected picture vector* are optional and specify values of ImageIndex and SelImageIndex respectively for each of the new items.

The new items are inserted with the first one being placed at the same level in the hierarchy as the item specified in element [1].

The result is an integer that reports the index position at which the first of the new items has been inserted.

AddRow   Event 152

Applies to   Grid

If enabled, this event is reported by the Grid object if the user presses the Cursor Down key, and the current cell (CurCell) is within the last row on the Grid. The default action is to append a new row to the contents of the Grid. If you attach a callback function to this event and have it return a value of 0, a new row will not be appended to the Grid. Note that the event will not be generated unless the first element of the AutoExpand property is set to 1.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 3 element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	'*AddRow*' or 152
[3] Row number:	integer

An application may insert a new row into a Grid by calling AddRow as a method. The argument is a 1 to 7-element array as follows:

[1] Row number:	integer
[2] Row title:	character vector or matrix
[3] Row height:	integer
[4] Undo flag:	0 or 1
[5] Resize flag	0 or 1
[6] Title colour:	negative scalar integer or 3-element RBG
[7] Line type:	integer

If you are using default row titles, *Row title* will be ignored and the rows will be re-labelled with default titles. If you have set RowTitles, the title you specify will be inserted. If you omit *Row title*, a blank title will be inserted.

Similarly, if you have not previously set CellHeights, ResizeRows, RowTitleFCol or RowLineTypes or if you have given them a scalar value, the corresponding parameter will be ignored. However, if you have specified CellHeights, ResizeRows, RowTitleFCol or RowLineTypes to be a *vector*, the number you specify in the corresponding parameter will be inserted into the appropriate property vector. If you omit *Row height*, it will be assigned a default value; new values for the other properties default to 0.

Undo flag (default 1) specifies whether or not the addition of the new row may subsequently be undone by an Undo event.

To insert a new row before the first one, you must specify the *Row number* as 1 (or 0 if `⎕IO` is 0). To add a new row after the last one, you may specify any number greater than the current number of rows. The data in the new row will be set to 0 if the Values property is numeric, or to an empty character vector otherwise.

Align   Property

Applies to   Animation, Button, CoolBar, DateTimePicker, ListView, Menu, MenuItem, Scroll, Spinner, Splitter, StatusBar, StatusField, TabBar, TabBtn, TabControl, ToolBar, ToolControl

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

For a Button, Menu, or MenuItem the Align property may be `'None'`, `'Left'` or `'Right'`. If the Button Style is `'Radio'` or `'Check'` this property specifies the position of the text relative to the button symbol. The default is `'Right'`. For a Button with Style `'Push'`, the value of Align is `'None'`.

For a Button with Style `'Radio'` or `'Check'` that is created as a child of a Grid the value of the Align property may also be `'Centre'` or `'Center'`. Either of these values causes the symbol part of the Button (the circle or checkbox) to be centred within the corresponding Grid cell(s).

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

For a Menu , MenuItem, or StatusField, Align `'Right'` is used to position the object at the right end of its parent MenuBar or StatusBar. `'None'` is equivalent to `'Left'` which is the default.

For objects of type CoolBar, Splitter, Scroll, StatusBar, TabBar, ToolBar and ToolControl, Align may be `'None'`, `'Top'`, `'Bottom'`, `'Left'` or `'Right'`. It specifies to which (if any) of the four sides of the parent the object is anchored and also the default position and size of the object. Specifying Align typically causes the Attach property to be set to appropriate values as follows :

Align	Attach
`'Top'`	`'Top' 'Left' 'Top' 'Right'`
`'Bottom'`	`'Bottom' 'Left' 'Bottom' 'Right'`
`'Left'`	`'Top' 'Left' 'Bottom' 'Left'`
`'Right'`	`'Top' 'Right' 'Bottom' 'Right'`

These settings cause the object to remain at a fixed distance (in pixels) from the corresponding edge of the parent. Furthermore, the object will have a fixed height or width, but its length will stretch and shrink as the Form is resized.

Note that this does not apply to a TabControl for which the default value of Attach is `'None'` `'None'` `'None'` `'None'`, regardless of the value of Align.

The default value of Align is `'Right'` for a vertical Scroll, `'Bottom'` for a horizontal Scroll, and `'Top'` for a CoolBar, TabBar, TabControl, ToolBar and ToolControl. Furthermore, unless Posn and Size are specified explicitly, the object is placed along the corresponding edge of its parent.

For a Scroll object, Align also determines the direction of a Scroll object unless it is overridden by setting HScroll or VScroll directly. If neither HScroll or VScroll is defined and Align is `'Top'` or `'Bottom'`, a horizontal scrollbar is provided. If neither HScroll or VScroll is defined and Align is `'None'`, `'Left'` or `'Right'`, a vertical scrollbar is provided.

The value of the Align property may only be assigned by `⎕WC` and may not be changed using `⎕WS`.

AlignChar   Property

Applies to   Grid

The AlignChar property specifies a character on which the data displayed in a column of a Grid is to be aligned vertically. It is useful to align columns of numbers that are formatted by the FormatString property. AlignChar may be a scalar or singleton that applies to all columns of the Grid.

If the data in the column is left-justified, it is aligned using the first occurrence of the alignment character in each cell counting from the left. If the data is right-justified, it is aligned using the first occurrence of the alignment character from the right-hand end of the text.

If the text in a cell does not contain an alignment character, it is aligned as if the alignment character were placed following the last digit.

AlphaBlend   Property

Applies to   Form

Windows 2000 only

The AlphaBlend property specifies a level of translucency which allows the area behind a Form to show through.

AlphaBlend is a scalar integer value in the range 0 to 255.

A value of 255 (the default) specifies no translucency, and the Form is entirely opaque obliterating anything behind it.

A value of 0 specifies total translucency and the Form itself is not visible. Furthermore, mouse events over the Form will not be reported by the Form itself but will be passed to any other windows underneath the Form.

Values in between specify varying levels of translucency.

AlphaBlend is effective only under Windows 2000 and later, and is ignored in earlier versions of Windows.

AmbientChanged Event 533

Applies to ActiveXContainer, ActiveXControl

If enabled, this event is reported when any of the ambient properties change in an application hosting an ActiveXControl object. The new values of the ambient properties are available from the FontObj, FCol and BCol properties of the ActiveXContainer.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to ¯1 or by returning 0 from a callback function.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1] Object name		character vector
[2] Event name or code		*'AmbientChanged'* or 533
[3] Property code		integer
[4] Description		character vector

For properties supported by Dyalog APL, Property code and Description may be one of the following:

Property Code	Description	Meaning
¯701	DISPID_AMBIENT_BACKCOLOR	BCol has changed
¯703	DISPID_AMBIENT_FORECOLOR	FCol has changed
¯705	DISPID_AMBIENT_FONT	Font has changed
¯1	DISPID_AMBIENT_UNKNOWN	unknown

Note that other ambient properties may be reported, although these have no corresponding Dyalog APL property.

Animation   Object

Purpose	The Animation object plays simple animations from AVI files and resources.
Parents	ActiveXControl, Form, Group, PropertyPage, SubForm
Children	Bitmap, Circle, Cursor, Ellipse, Font, Marker, Poly, Rect, Text, Timer
Properties	Type, Posn, Size, File, Coord, Border, Active, Visible, Event, Sizeable, Dragable, BCol, AutoConf, Data, Attach, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, Translate, AcceptFiles, KeepOnClose, AutoPlay, Transparent, Align, MethodList, ChildList, EventList, PropList
Events	AnimStarted, AnimStopped, Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, GotFocus, Help, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel
Methods	Animate, AnimClose, AnimOpen, AnimPlay, AnimStop, Detach, GetFocus, GetTextSize, ShowSIP

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

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

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

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

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

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

The AnimStop method causes the animation to stop.

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

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

Animate   Method 29

Applies to   ActiveXControl, Animation, Button, Calendar, ColorButton, Combo, ComboEx, CoolBar, DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, ProgressBar, PropertyPage, RichEdit, Scroll, SM, Spinner, Static, StatusBar, SubForm, TabBar, TabControl, ToolBar, ToolControl, TrackBar, TreeView, UpDown

Windows 2000 only

The Animate method enables you to produce special effects when showing or hiding objects. There are three types of animation: roll, slide, and alpha-blended fade.

The argument to Animate is a 1 or 2-element array as follows:

[1] Effects:		integer
[2] Play time:		integer

The value of the *Effects* parameter is the sum of the following flags:

Flag	Value	Description
AW_HOR_POSITIVE	1	Animates the window from left to right. This flag can be used with roll or slide animation. It is ignored when used with the AW_CENTER flag.
AW_HOR_NEGATIVE	2	Animates the window from right to left. This flag can be used with roll or slide animation. It is ignored when used with the AW_CENTER flag.
AW_VER_POSITIVE	4	Animates the window from top to bottom. This flag can be used with roll or slide animation. It is ignored when used with the AW_CENTER flag.
AW_VER_NEGATIVE	8	Animates the window from bottom to top. This flag can be used with roll or slide animation. It is ignored when used with the AW_CENTER flag.
AW_CENTER	16	Makes the window appear to collapse inward if being hidden or expand outward if being displayed
AW_SLIDE	262144	Uses slide animation. By default, roll animation is used. This flag is meaningless on its own but is ignored when used with the AW_CENTER flag.
AW_BLEND	524288	Uses a fade effect. This flag can be used only for a Form.

The Playtime parameter is optional and specifies the length of time over which the animation is played in milliseconds. The default value depends upon the animation but is typically 200 milliseconds.

AnimClose   Method 291

Applies to   Animation

The AnimClose method closes the AVI file that is currently loaded in an Animation object. The display is reset to the object's background colour.

AnimClose is niladic.

AnimOpen   Method 290

Applies to   Animation

The AnimOpen method opens an AVI file in an Animation object.

The argument to AnimOpen is a 1 or 2-element array as follows:

> [1] File: character vector

> [2] Resource id: integer

If a single element is specified, it represents the name of a .AVI file.

If 2 elements are specified, the first element specifies the name of a DLL or EXE and the second element identifies the particular AVI resource stored in that file. The identifier may be its name (a character string) or its resource id (a non-zero positive integer).

If the AutoPlay property is set to 1, the animation will play immediately. Otherwise, only the first frame will be displayed.

Note that the Animation object can only play AVI files or resources that have no sound and can only display uncompressed AVI files or AVI files that have been compressed using Run-Length Encoding (RLE). If you attempt to open an inappropriate AVI file, the operation will fail with a DOMAIN ERROR and the following message will be displayed in the Status Window:

AVI file includes sound data or is in a format not supported by the Animation object

AnimPlay   Method 292

Applies to   Animation

The AnimPlay method plays an AVI clip in an Animation object.

The argument to AnimPlay is a 3-element array as follows:

[1] Repeat:		integer
[2] From:		integer
[3] To:		integer

Repeat specifies the number of times the clip is repeated. A value of -1 causes the clip to be repeated indefinitely.

From is a 0-based index of the frame where playing begins and must be less than 65536. A value of zero means begin with the first frame in the AVI clip

To is a 0-based index of the frame where playing ends and must be less than 65536. A value of -1 means end with the last frame in the AVI clip

The last frame remains displayed until the clip is unloaded using AnimClose or until another clip is loaded.

AnimStarted Event 294

Applies to Animation

If enabled, this event is reported by an Animation object just before an AVI clip starts playing

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	`'AnimStarted'` or 294

This event is reported for information only and cannot be disabled or modified in any way.

AnimStop Method 293

Applies to Animation

The AnimStop method stops playing an AVI clip in an Animation object.

AnimStop is niladic.

The last frame remains displayed until the clip is unloaded using AnimClose or until another clip is loaded.

AnimStopped	Event 295

Applies to   Animation

If enabled, this event is reported by an Animation object just after an AVI clip has stopped playing

The event message reported as the result of ⎕*DQ*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1] Object: ref or character vector

[2] Event name or code: '*AnimStopped*' or 295

This event is reported for information only and cannot be disabled or modified in any way.

APLVersion	Property

Applies to   Root

This is a read-only property that provides information about the Version of Dyalog APL that you are using. It is a 4-element vector of character vectors containing:

[1]   the target environment

[2]   the version

[3]   the version type ('*W*'=Windows, '*M*'=Motif, '*P*'=Pocket APL)

[4]   '*Development*' or '*Runtime*' or '*DLL*'

Example

```
      DISPLAY '.' ⎕WG'APLVersion'
.→--------------------------------------.
| .→------. .→-----. .→. .→----------. |
| |Windows| |10.0.1| |W| |Development| |
| '-------' '------' '-' '-----------' |
'∈--------------------------------------'
```

ArcMode Property

Applies to   Circle, Ellipse

This property determines how arcs are drawn. Its value is 0, 1 or 2.

0	:	only the arc is drawn
1	:	arcs define "arc segments", with a single straight line joining the two ends of the arc together
2	:	arcs define "pie segments", with lines drawn from the start and end points of the arc to the centre

Note that the segments defined by ArcMode 1 and 2 may be filled (by setting FStyle).

Array Property

Applies to   Clipboard

This property may be used to set or retrieve the contents of the Windows clipboard as a Dyalog APL array.

Attach   Property

Applies to   Animation, Button, Calendar, ColorButton, Combo, ComboEx,
CoolBar, DateTimePicker, Edit, Grid, Group, Label, List, ListView,
MDIClient, ProgressBar, RichEdit, Scroll, SM, Spinner, Static,
StatusBar, StatusField, SubForm, TabBar, TabBtn, TabControl,
ToolBar, ToolControl, TrackBar, TreeView, UpDown

This property specifies how an object responds to its parent being resized. It is a 4-element vector of character vectors which defines how each of the four edges of the object moves in response to a resize request made by the parent. Note that this property is only effective if the value of AutoConf on the parent is 2 or 3 and AutoConf for the object itself is 1 or 3.

The 4 elements of Attach refer to the Top, Left, Bottom and Right edges of the object respectively. Their values may be defined as follows :

Element	Value	Meaning
[1]	`'Top'`	The top edge of the object is attached to the top edge of its parent.
	`'Bottom'`	The top edge of the object is attached to the bottom edge of its parent.
	`'None'`	The top edge of the object is not attached to its parent.
[2]	`'Left'`	The left edge of the object is attached to the left edge of its parent.
	`'Right'`	The left edge of the object is attached to the right edge of its parent
	`'None'`	The left edge of the object is not attached to its parent.

Element	**Value**	**Meaning**
[3]	*'Top'*	The bottom edge of the object is attached to the *top* edge of its parent.
	'Bottom'	The bottom edge of the object is attached to the *bottom* edge of its parent.
	'None'	The bottom edge of the object is not attached to its parent.
[4]	*'Left'*	The right edge of the object is attached to the *left* edge of its parent.
	'Right'	The right edge of the object is attached to the *right* edge of its parent.
	'None'	The right edge of the object is not attached to its parent.

If an edge of the object is attached to an edge of its parent, its position in absolute (pixel) terms remains fixed relative to that edge when its parent is resized. Thus if Coord is *'Pixel'*, the corresponding Posn or Size property of the object remains unaffected by the resize. If Coord has any other value, the value of Posn or Size will change.

If an edge of the object is *not* attached to its parent, its absolute position (in pixels) will change in proportion to the size change (in the corresponding direction) of its parent. Thus if Coord is *'Pixel'*, the corresponding Posn or Size property of the object will change as a result of the resize. If Coord has any other value, the value of Posn or Size will be unaffected.

The default value of Attach is (*'None' 'None' 'None' 'None'*). This causes the object to reposition and resize itself in proportion to its parent.

Some objects have an Align property which, among other things, provides a quick way to set their Attach property. Examining this mechanism may help to further explain how the Attach property works. Setting Align to `'Top'` has the effect of setting Attach to (`'Top'` `'Left'` `'Top'` `'Right'`). Attaching the top edge of the object to the top edge of its parent causes the object to remain at a fixed distance from the top edge of its parent. The additional measure of attaching its bottom edge to the top edge of its parent causes the height of the object to remain fixed. Attaching the left and right edges of the object to the corresponding edges of its parent causes the object to shrink and expand as the parent is resized horizontally. If you position the object at (0 0) and set its width to be the same as the width of its parent, you have an object that always occupies the entire length of its parent, yet remains of fixed height. This is precisely the behaviour required for a ToolBar or a top Scroll. For further details, see Align property.

The illustration below shows a Form containing a Static. The subsequent four illustrations show the result after the Form has been resized, using different settings for the Attach property of the Static.

Attach is (Top None Top None)

Attach is (None Left None Left)

Attach is (Top Left Bottom Right)
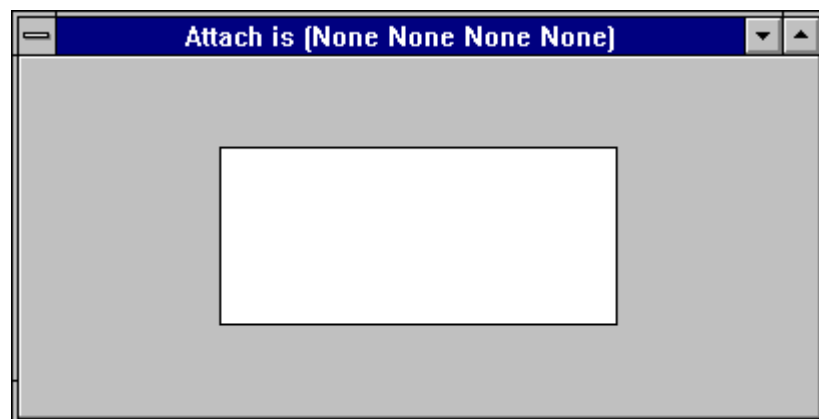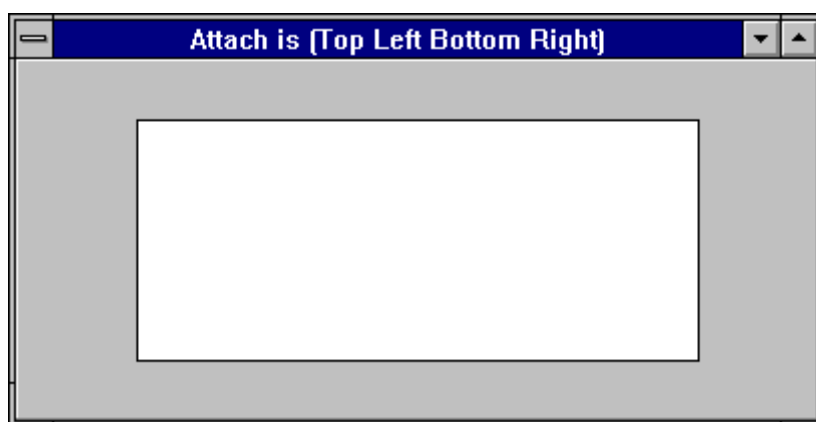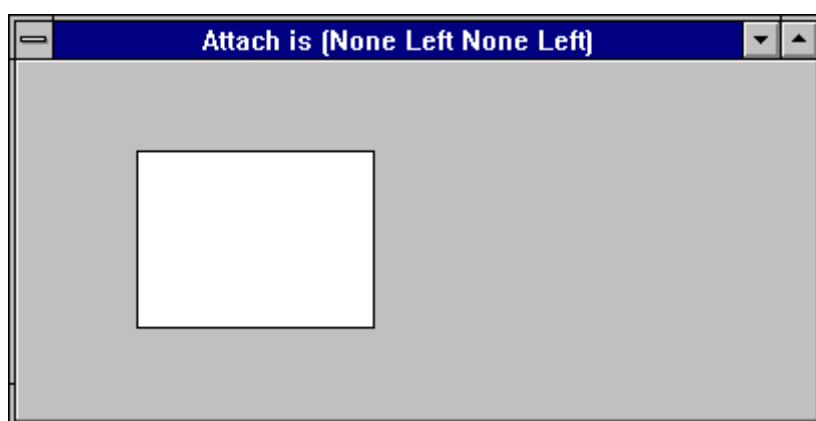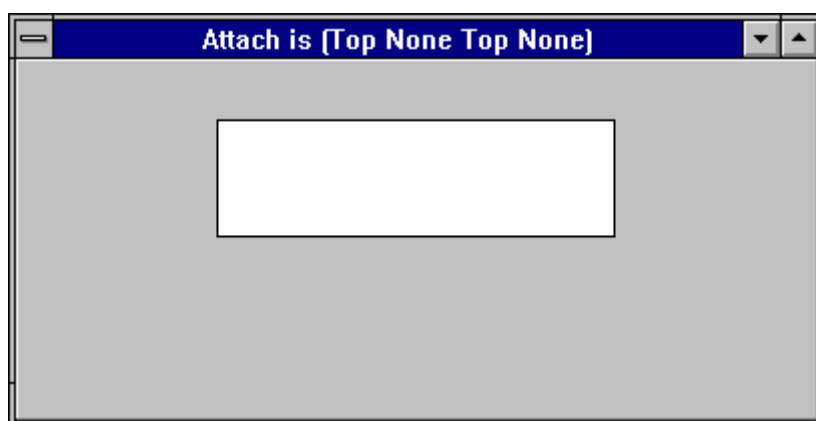
AutoArrange   Property

Applies to   ListView

The AutoArrange property is Boolean and specifies whether or not the items in a ListView object are automatically re-arranged when a single item is repositioned. Its default value is 0.

AutoBrowse   Property

Applies to   OLEClient

This property is retained for backwards compatibility with previous versions of Dyalog APL, but is no longer relevant. Setting it has no effect.

AutoConf   Property

Applies to   ActiveXControl, Animation, Button, Calendar, Circle, ColorButton, Combo, ComboEx, DateTimePicker, Edit, Ellipse, Form, Grid, Group, Image, Label, List, ListView, Marker, Poly, ProgressBar, Rect, RichEdit, Scroll, SM, Spinner, Static, StatusBar, StatusField, SubForm, TabBar, TabBtn, Text, ToolBar, TrackBar, TreeView, UpDown

This property determines what happens to an object when its **parent** is resized, and how resizing an object affects its children. It may take one of the following values; the default is 3.

0	:	Ignore resize by parent. Do not propagate resize to children.
1	:	Accept resize by parent. Do not propagate resize to children.
2	:	Ignore resize by parent. Do propagate resize to children.
3	:	Accept resize by parent. Do propagate resize to children.

If AutoConf is 0 or 2, the object's **physical** size (in pixels) and position (in pixels) relative to the top left corner of its parent remains unchanged when its parent is resized.

If the object has `'Prop'` or `'User'` co-ordinates, the values of its Posn and Size properties will change as a result.

If AutoConf is 1 or 3, by default the object is physically reconfigured when its parent is resized, such that its **relative** size and position within its parent remain unchanged. If the object has '*Pixel*' co-ordinates, the values of its Posn and Size properties will change as a result. Note that this default processing can be prevented by inhibiting the Configure (31) Event (see below).

If AutoConf is 0 or 1 and the object is resized, either by its parent or directly by the user, it does **not** attempt to physically reconfigure its children. This means that if the children have '*Prop*' or '*User*' co-ordinates, the values of their Posn and Size co-ordinates will change as a result.

If AutoConf is 2 or 3 and the object is resized, either by its parent or directly by the user, it propagates a Configure (31) Event to each of its children. By default this means that the object's children will be physically reconfigured so that they maintain their relative positions and sizes within it. If their co-ordinate system is '*Pixel*', the values of their Posn and Size properties will change as a result.

Note that additional or alternative control may be imposed by inhibiting the Configure (31) Event. This can be done either by setting the event's "action" code to ¯1 or by returning a 0 from a callback function attached to it.

AutoExpand   Property

Applies to   Grid

This property is a 2-element Boolean value that specifies whether or not rows and columns may be added to a Grid object by the user.

If the first element of AutoExpand is 1, a row is added when the current cell is within the last row of the Grid and the user presses Cursor Down. Similarly, if the second element is 1, a column is added when the current cell is within the last column of the Grid and the user presses Cursor Right. The default value for AutoExpand is (0 0).

Note that when a row or column is added, the appropriate properties (including Values and CellTypes)are expanded accordingly.

If AutoExpand is enabled, the Grid generates AddRow and AddCol events. You can return a zero from a callback function to selectively prevent the addition of rows and columns if appropriate.

AutoPlay   Property

Applies to   Animation

Specifies whether or not an AVI clip is played immediately when loaded in an Animation object.

AutoPlay is a single number with the value 0 (the default) or 1. If AutoPlay is 1, the AVI clip is automatically played through once from beginning to end when loaded from a file by the AnimOpen method.

BadValue   Event 180

Applies to   Edit, Spinner

If enabled, this event is reported by an Edit object whose FieldType property is set when the user enters invalid data into the object and then switches focus to another control or to another application. The default action of the event is to sound the bell (beep). You can disable this action by returning 0 from a callback function or by setting its action code to ¯1. Note that in neither case is the Value property of the object updated. The event message reported as the result of ⎕DQ, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	'*BadValue*' or 180
[3] Object:	ref or character vector

The third element of the event message is either the name of the control to which the user has switched the focus, or is an empty vector if the focus has gone to another application.

BandBorders   Property

Applies to   CoolBar

The BandBorders property specifies whether or not narrow lines are drawn to separate adjacent bands in a CoolBar.

BandBorders is a single number with the value 0 (no lines) or 1 (lines are displayed); the default is 0.

BCol   Property

Applies to   ActiveXContainer, ActiveXControl, Animation, Button, Circle,
                    Combo, ComboEx, CoolBand, CoolBar, Edit, Ellipse, Form, Grid,
                    Group, Label, List, ListView, MDIClient, Menu, MenuItem, Poly,
                    ProgressBar, Rect, RichEdit, Scroll, Separator, SM, Spinner, Splitter,
                    Static, StatusBar, StatusField, SubForm, TabBar, TabBtn, Text,
                    TipField, ToolBar, TrackBar, TreeView, UpDown

This property defines the background colour(s) of an object. For objects with more than
one constituent part, it may specify a set of background colours, one for each part. A
single colour is represented by a single number which refers to a standard colour, or by a
3-element vector which defines a colour explicitly in terms of its red, green and blue
intensities.

If BCol is 0 (which is the default) the background colour is defined by your current
colour scheme for the object in question. For example, if you select yellow as your
Windows "Menu Bar" colour, you will by default get a yellow background in Menu and
MenuItem objects, simply by not specifying BCol or by setting it to 0.

A negative value of BCol refers to a standard Windows colour as described below.
Positive values are reserved for a possible future extension.

BCol	Colour Element	BCol	Colour Element
0	Default	¯11	Active Border
¯1	Scroll Bars	¯12	Inactive Border
¯2	Desktop	¯13	Application Workspace
¯3	Active Title Bar	¯14	Highlight
¯4	Inactive Title Bar	¯15	Highlighted Text
¯5	Menu Bar	¯16	Button Face
¯6	Window Background	¯17	Button Shadow
¯7	Window Frame	¯18	Disabled Text
¯8	Menu Text	¯19	Button Text
¯9	Window Text	¯20	Inactive Title Bar Text
¯10	Active Title Bar Text	¯21	Button Highlight

If BCol is set to θ (zilde), the object is drawn with a transparent background.

If BCol contains a 3-element vector, it specifies the intensity of the red, green and blue
components of the colour as values in the range 0-255. For example, (255 0 0) is red and
(255 255 0) is yellow. Note however that the colour realised depends upon the
capabilities of the display adapter and driver.

For a Button, BCol is only effective if the Style is `'Radio'` or `'Check'` and is ignored if the Style is `'Push'`.

It is recommended that you only use **pure** background colours in Combo and Edit objects. This is because the text written in these objects cannot itself have a dithered background.

For the Ellipse, Poly and Rect objects, BCol specifies the background colour of the line drawn around the perimeter of the object and is effective only when a non-solid line (LStyle 1-4) is used. It also specifies the colour used to fill the spaces between hatch lines if a hatch fill (FStyle 1-6) is used.

BeginEditLabel   Event 300

Applies to   ListView, TreeView

If enabled, this event is reported when the user clicks on an item in a ListView or TreeView object that has the focus, and signals the start of an edit operation. The default processing for the event is to display a pop-up edit box around the item and to permit the user to change its text.

You may disable the operation by setting the action code for the event to ¯1. You may prevent a particular item from being edited by returning 0 from a callback function. You may also initiate the edit operation by calling BeginEditLabel as a method.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	`'BeginEditLabel'` or 300
[3] Item number:	Integer. The index of the item.

Bitmap   Object

Purpose	A graphical object used to represent a bitmap which may be used both to display a picture or as a pattern (brush) used to fill other objects.
Parents	ActiveXControl, Animation, Button, CoolBand, Form, Grid, Group, ImageList, ListView, Menu, MenuBar, MenuItem, NetType, OLEServer, Printer, ProgressBar, PropertyPage, PropertySheet, RichEdit, Root, StatusBar, SubForm, TCPSocket, ToolBar, ToolButton, ToolControl, TrackBar, TreeView, UpDown
Children	Circle, Ellipse, Font, Image, Marker, Metafile, Poly, Rect, Text, Timer
Properties	Type, File, Bits, CMap, KeepBits, Size, Coord, Event, FontObj, YRange, XRange, Data, TextSize, Translate, Accelerator, KeepOnClose, CBits, MaskCol, MethodList, ChildList, EventList, PropList
Events	Close, Create, Select
Methods	Detach, FileRead, FileWrite, GetTextSize, MakeGIF, MakePNG

A Bitmap may be created either from a file (.BMP) or from APL arrays. To create a Bitmap object using `⎕WC`, you can either specify the File property **or** the CBits property, **or** the Bits and CMap properties.

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

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

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

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

The KeepBits property has the value 0 or 1, and controls how a Bitmap is saved in the workspace.

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

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

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

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

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

> `bmname.FileWrite`

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

> `bmname.FileRead`

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

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

Using a bitmap is always a 2-stage process. First you create a Bitmap object with `⎕WC`. Then you use it by specifying its name as a property of another object.

The Picture property specifies the name of a Bitmap to be displayed in an ActiveXControl, Button, Form, Group, Image, MDIClient, SM, Static, StatusBar, StatusField, SubForm, TabBar, or ToolBar. The BtnPix property specifies the names of three Bitmaps to be used to represent the 3 states of a Button, Menu or MenuItem. The FStyle property specifies the name of a Bitmap to be used as a pattern to fill a Poly, Ellipse or Rect object.

Bits   Property

Applies to   Bitmap, Clipboard, Cursor, Icon

This property defines the pattern in a Bitmap, Cursor, or Icon object, or the pattern of a bitmap stored in the Windows clipboard.

For a Bitmap, Clipboard or Icon, Bits is an integer matrix each of whose elements represents the colour of the corresponding pixel in the bitmap. The colours are specified as 0-origin indices into the CMap property, which itself defines the complete set of different colours (the colour map) used by the object. See CMap for further details.

Please note that Bits and CMap may **only** be used to represent an image with a colour palette of **256 colours or less**. If the colour palette is larger, the values of Bits and CMap reported by □*WG* will be (0 0). For a high-colour image, use CBits instead.

For a Cursor, Bits is a Boolean matrix which specifies the shape of the cursor. For a Cursor and Icon, Bits is used in conjunction with the Mask property.

Border   Property

Applies to   ActiveXControl, Animation, Button, Calendar, Combo, ComboEx, DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, RichEdit, Scroll, SM, Spinner, Static, StatusBar, StatusField, SubForm, TabBtn, ToolBar, TrackBar, TreeView, UpDown

This property specifies whether or not an object is displayed with a border around it. The value of Border may only be set by □*WC*. It is a single number with the value 0 (no border), 1 (border) or 2. The value 2 applies only to a Form and is used in combination with (*'EdgeStyle' 'Dialog'*) to obtain standard dialog box appearance

For a Form or SubForm, the value of the Border property is only relevant if Sizeable, Moveable, SysMenu, MaxButton and MinButton are **all** 0.

Browse   Method 585

Applies to   OCXClass, OLEClient

The Browse method causes APL to browse a type library and to create methods and properties in an OCXClass or OLEClient namespace that correspond to each of the methods and properties described in the type library.

If there is no type information automatically associated with a COM object, the OLEClient or OCXClass namespace created by $\Box WC$ will not contain any of the methods and properties that should be exported by the object. Browse is used to correct this situation manually.

This method is supported by all OLEClient objects, i.e. the top-level OLEClient that is associated with the OLE object, and all the namespaces associated with any sub-objects exposed as the result of methods, properties or events.

The argument is θ, a simple character vector, or a 1 or 2-element array as follows:

      [1] Interface name: character vector

      [2] File name: character vector

Interface name is the name or CLSID of the interface

File name is he name of the type library file to be browsed.

If called with an argument of θ, Browse attempts to use the type library that is associated with the object (if any).

BrowseBox Object

Purpose	The BrowseBox object allows the user to browse for and select a folder or other resource.
Parents	ActiveXControl, CoolBand, Form, Grid, OLEServer, PropertyPage, PropertySheet, Root, StatusBar, SubForm, TCPSocket, ToolBar, ToolControl
Children	Timer
Properties	Type, Caption, BrowseFor, Target, StartIn, HasEdit, Event, Data, Translate, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, FileBoxCancel, FileBoxOK
Methods	Detach, Wait

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

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

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

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

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

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

Example:

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

BrowseFor   Property

Applies to   BrowseBox

The BrowseFor property is a character vector that specifies the type of resource to be the target of a BrowseBox object.

BrowseFor may be *'Directory'* (the default), *'File'*, *'Computer'* or *'Printer'*.

BtnPix   Property

Applies to   Button, Menu, MenuItem

This property is used to customise the appearance of a Button, Menu or MenuItem. It specifies the names of up to 3 Bitmap objects to be used to display the object under different circumstances. In general, BtnPix is a 3-element vector of character vectors. However, if it defines a single Bitmap, it may be a simple character scalar or vector, or an enclosed character vector.

The first Bitmap is displayed when the object is shown in its normal state. For a Button, this is when its State is 0. The second Bitmap is used for a Menu or MenuItem, when the object is selected (highlighted), or for a Button when its State is 1. The third Bitmap is used when the object is disabled by having its Active property set to 0.

For a Button with Style *'Push'*, this means that when the user clicks the Button, its appearance switches from the first to the second Bitmap, and then back again. To maintain the standard 3-D appearance, the Bitmaps should contain the correct shadow lines around their edges. For Buttons with Style *'Radio'* or *'Check'*, the Button will display one or other of the two Bitmaps according to its current State.

For example, to have a Button that displays a "Tick" or a "Cross" according to its State :

```
'YES' □WC 'Bitmap' 'C:\DYALOG82\YES.BMP'
'NO'  □WC 'Bitmap' 'C:\DYALOG82\NO.BMP'

'f1.r1' □WC 'Button'('Style' 'Check')
                    ('BtnPix' 'YES' 'NO')
```

Btns   Property

Applies to   MsgBox

The Btns property determines the set of buttons to be displayed in a MsgBox. It is a simple vector (one button) or a matrix with up to 3 rows, or a vector of up to 3 character vectors specifying the captions for up to 3 buttons. The buttons are arranged along the bottom of the dialog box in the order specified.

Under Windows, there are restrictions on these buttons. However the property has been designed more generally to be useful under different GUIs. Under Windows, the Btns property may specify one of six sets of buttons as follows.

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

If any other combination is specified, *⎕WC* and *⎕WS* will report a *DOMAIN ERROR*. The names of the buttons are however case-insensitive, so the system will accept *'ok'*, *'Ok'*, *'oK'* or *'OK'*.

If the Btns property is not specified, it assumes a default according to Style as follows :

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

If Style is not specified, Btns defaults to *'OK'*.

Button   Object

Purpose	Allows the user to initiate an action or to select an option using a button.
Parents	ActiveXControl, CoolBand, Form, Grid, Group, PropertyPage, SubForm, ToolBar, ToolControl
Children	Bitmap, Circle, Cursor, Ellipse, Font, Marker, Poly, Rect, Text, Timer
Properties	Type, Caption, Posn, Size, Style, Coord, Align, State, Default, Cancel, Border, Justify, Active, Visible, Event, Sizeable, Dragable, FontObj, FCol, BCol, Picture, BtnPix, CursorObj, AutoConf, Data, Attach, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, ReadOnly, Translate, Accelerator, AcceptFiles, KeepOnClose, Redraw, MethodList, ChildList, EventList, PropList
Events	Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, FontCancel, FontOK, GotFocus, Help, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Select
Methods	Animate, ChooseFont, Detach, GetFocus, GetTextSize, ShowSIP

The type of button displayed is determined by the Style property which may take the value `'Push'`, `'Radio'`, `'Check'` or `'Toggle'`. Under Windows, `'Toggle'` and `'Check'` are treated identically.
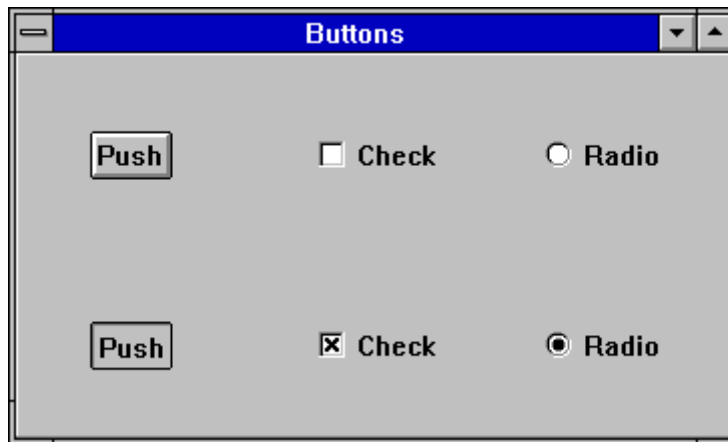
Differing values of Style with State 0 and 1

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

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

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

Radio and Check buttons **also** generate Select events when their State changes, and you can attach callback functions to these events to keep track of their settings. However, as Radio and Check buttons are not normally used to generate actions, it is perhaps easier to wait until the user signifies completion of the dialog box in some way, and then query the State of the buttons using $\square WG$. For example, if you have a set of Radio or Check buttons in a Group called $f1.options$, the following statements retrieve their settings.

> $OPTIONS \leftarrow (\square WN \ 'f1.options') \ \square WG¨⊂'State'$

or

> $OPTIONS \leftarrow \square WG∘'State' \ ¨ \ \square WN \ 'f1.options'$

The Caption property determines the text displayed in the Button. Its default value is an empty vector. If Style is $'Radio'$ or $'Check'$, the text may be aligned to the left or right of the button graphic using the Align property. Its default value is $'Right'$.

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

If Size is not specified, the size of the button is determined by its Caption. If either element of Size is set to 0 the corresponding dimension is determined by the height or width of its Caption. If Caption is not specified, or is set to an empty vector, the value of Size is set to a default value.

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

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

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

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

Calendar Object

Purpose	The Calendar object provides an interface to the Month Calendar Control.
Parents	ActiveXControl, Form, Group, PropertyPage, SubForm, ToolBar
Children	Cursor, Font, Menu, MsgBox, TCPSocket, Timer
Properties	Type, Posn, Size, Style, Coord, Border, Active, Visible, Event, FirstDay, MaxSelCount, SelDate, MinDate, MaxDate, CalendarCols, Today, HasToday, CircleToday, WeekNumbers, MonthDelta, Sizeable, Dragable, FontObj, CursorObj, AutoConf, Data, Attach, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, Accelerator, KeepOnClose, Redraw, MethodList, ChildList, EventList, PropList
Events	CalendarDblClick, CalendarDown, CalendarMove, CalendarUp, Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, FontCancel, FontOK, GetDayStates, GotFocus, Help, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, SelDateChange, Select
Methods	Animate, ChooseFont, DateToIDN, Detach, GetFocus, GetMinSize, GetTextSize, GetVisibleRange, IDNToDate, ShowSIP

The Calendar object displays a calendar and allows the user to select a date or range of dates. The following illustration shows a default Calendar object.
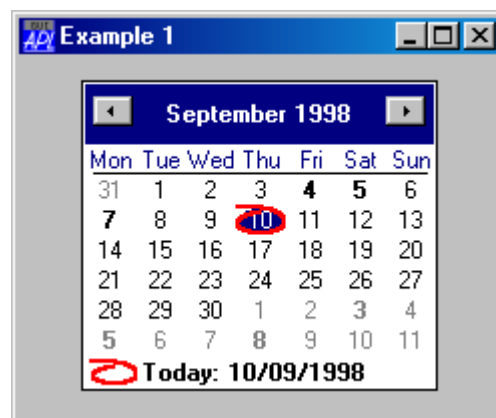
The Calendar object requires the Windows Custom Control Library COMCTL32.DLL, Version 4.70 or higher.

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

The Today property is an IDN that specifies the current day. Its default value is today's date, i.e. the local date set on your computer. An IDN is an integer that represents a date as the number of days starting at Jan 1st 1900, i.e. Jan 01 1900 is IDN 1. Note that IDNs calculated by Dyalog APL do not precisely match IDNs calculated by Microsoft Excel which incorrectly assumes that 1900 was a Leap Year.

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

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

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

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

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

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

The Style property may be either `'Single'` (the default) or `'Multi'`. If Style is `'Single'`, the user may select a single date. If Style is `'Multi'`, the user may select a contiguous range of dates. In this case, the maximum number of contiguous days that can be selected is defined by the MaxSelCount property which is an integer whose default value is 7.

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

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

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

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

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

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

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

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

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

CalendarCols   Property

Applies to   Calendar, DateTimePicker

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

CalendarCols is a 6-element integer vector whose elements specify the colours as follows:

[1]   Background colour displayed between months

[2]   Background colour displayed within the month.

[3]   Text colour within a month

[4]   Background colour displayed in the calendar's title

[5]   Colour used to display text within the calendar's title

[6]   Colour used to display header day and trailing day text. Header and trailing days are the days from the previous and following months that appear on the current month calendar.

Each element of CalendarCols may be 0 (which means default colour), a negative singleton that specifies a particular Windows colour, or a 3-element integer vector of RGB values.

Note: At the time of writing, setting the first element of CalendarCols has no effect. Dyadic believes this to be a Windows problem that may be corrected in due course.

CalendarDblClick   Event 273

Applies to   Calendar

If enabled, this event is reported when the user double-clicks the left mouse button over a Calendar object.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to ¯1 or by returning 0 from a callback function.

The event message reported as the result of $\Box DQ$, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	*'CalendarDblClick'* or 273
[3] Item Number:	integer
[4] Mouse Button:	integer
[5] Shift State:	integer. Sum of 1=shift key, 2=ctrl key, 4=alt key
[6] Element Type:	integer

For the meaning of elements 3 and 6, see CalendarDown.

CalendarDown Event 271

Applies to Calendar

If enabled, this event is reported when the user depresses the left mouse button over a Calendar object.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to ¯1 or by returning 0 from a callback function.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1] Object:		ref or character vector
[2] Event name or code:		*'CalendarDown'* or 271
[3] Item Number:		integer (see below)
[4] Mouse Button:		integer
[5] Shift State:		integer. Sum of 1=shift key, 2=ctrl key, 4=alt key
[6] Element Type:		integer (see below)

The 6th element of the event message is one of the following values:

Value	Description
0	NOWHERE The mouse pointer was not on the month calendar control, or it was in an inactive portion of the control.
1	CALENDARBK The mouse pointer was in the calendar's background.
2	CALENDARDATE The mouse pointer was on a particular date within the calendar
3	CALENDARDATENEXT The mouse pointer was over a date from the next month (partially displayed at the end of the currently displayed month). If the user clicks here, the month calendar will scroll its display to the next month or set of months.
4	CALENDARDATEPREV The mouse pointer was over a date from the previous month (partially displayed at the end of the currently displayed month). If the user clicks here, the month calendar will scroll its display to the previous month or set of months.
5	CALENDARDAY The mouse pointer was over a day abbreviation ("Fri", for example).
6	CALENDARWEEKNUM The mouse pointer was over a week number
7	TITLEBK The mouse pointer was over the background of a month's title.
8	TITLEBTNNEXT The mouse pointer was over the button at the top right corner of the control. If the user clicks here, the month calendar will scroll its display to the next month or set of months.
9	TITLEBTNPREV The mouse pointer was over the button at the top left corner of the control. If the user clicks here, the month calendar will scroll its display to the previous month or set of months.
10	TITLEMONTH The mouse pointer was in a month's title bar, over a month name.
11	TITLEYEAR The mouse pointer was in a month's title bar, over the year value.

If the value of the 6th element of the event message is 2 (CALENDARDATE), the 3rd element is the corresponding date reported as an IDN.

If the value of the 6th element of the event message is 5 (CALENDARDAY), the 3rd element is the index of the corresponding weekday (0-6).

If the value of the 6th element of the event message is 6 (CALENDARWEEKNUM), the 3rd element is the date of the first (leftmost) day in the corresponding week, reported as an IDN.

Otherwise, the 3rd element of the event message is 0.

CalendarMove   Event 274

Applies to   Calendar

If enabled, this event is reported when the user moves the left mouse button over a Calendar object.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to ¯1 or by returning 0 from a callback function.

The event message reported as the result of $\Box DQ$, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	*'CalendarMove'* or 274
[3] Item Number:	integer
[4] Mouse Button:	integer
[5] Shift State:	integer. Sum of 1=shift key, 2=ctrl key, 4=alt key
[6] Element Type:	integer

For the meaning of elements 3 and 6, see CalendarDown.

CalendarUp   Event 272

Applies to   Calendar

If enabled, this event is reported when the user releases the left mouse button over a Calendar object.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to ¯1 or by returning 0 from a callback function.

The event message reported as the result of *⎕DQ*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1] Object name		character vector
[2] Event name or code		*'CalendarUp'* or 272
[3] Item Number		integer
[4] Mouse Button		integer
[5] Shift State		integer. Sum of 1=shift key, 2=ctrl key, 4=alt key
[6] Element Type		integer

For the meaning of elements 3 and 6, see CalendarDown.

Cancel   Property

Applies to   Button

This property determines which (if any) Push button in a Form or SubForm is to be associated with the Escape key. It has the value 1 or 0.

Pressing the Escape key will generate a Select event on the Button whose Cancel property is 1, regardless of which object has the keyboard focus.

As only one button in a Form or SubForm can be the Cancel button, setting Cancel to 1 for a particular button automatically sets Cancel to 0 for all others in the same Form.

CancelToClose   Method 367

Applies to   PropertySheet

This method is used to change the buttons in a PropertySheet object. Its effect is to disable the Cancel button and, if the Style of the PropertySheet is *'Standard'*, it changes the text of the OK button to "Close". There is no result.

The CancelToClose method is niladic.

Caption   Property

Applies to   BrowseBox, Button, ColorButton, CoolBand, FileBox, Form, Group, Label, Menu, MenuItem, MsgBox, PropertyPage, PropertySheet, Root, StatusField, SubForm, TabBtn, TabButton, ToolButton

The Caption property is a character vector specifying fixed text associated with the object. For example, Caption defines the label on a Button, the title of a Form, SubForm or MsgBox, the heading in a Group, and the text of a Menu or a MenuItem.

For the Root object, Caption specifies the text displayed when Alt+Tab is used to switch to the Dyalog APL/W application. It may be used in conjunction with the IconObj property which specifies the name of an Icon object to be displayed alongside this text.

Its default value is an empty vector.

CaseSensitive   Property

Applies to   ComboEx

Specifies whether or not string searches in the items displayed by a ComboEx object will be case sensitive. Searching occurs when text is being typed into the edit box portion of the ComboEx

CBits   Property

Applies to   Bitmap, Clipboard

The CBits property represents the picture in a Bitmap object.

CBits provides an alternative representation to that provided by the Bits and CMap properties which apply only to Bitmaps with 256 colours or under. CBits may be used to represent both low-colour and high-colour bitmaps.

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

The colour value of each pixel is obtained by encoding the red, green and blue components, i.e.

```
PIXEL←256⊥RED GREEN BLUE
```

CellChange   Event 150

Applies to   Grid

If enabled, this event is reported when the user changes the contents of a cell in a Grid object and then attempts to move to another cell or to another control outside the Grid

The purpose of this event is to give the application the opportunity to perform additional validation before the update occurs (and to prevent it if necessary) or to update other cells in the Grid as a result of the change.

The default action for the CellChange event is to update the appropriate element of the Values property with the new data. This action can be disabled by returning 0 from the attached callback function. Notice however, that the user is not prevented from moving away from the cell. If you are using this event to perform additional validation and you require the user to correct the data before moving away, you must force the user back to the cell in question by generating a CellMove event.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is an 8-element vector as follows:

[1] Object:		ref or character vector
[2] Event code:		`'CellChange'` or 150
[3] Cell row:		integer
[4] Cell column:		integer
[5] New data:		number or character array
[6] New object:		ref or character vector (object to which the user has transferred focus)
[7] New cell row:		integer
[8] New cell column:		integer

If the user moves to another cell in the Grid, the 6th element of the event message is the name of the Grid object and elements 7 and 8 specify the new cell address ($\square IO$ dependent).

If the user switches the input focus to another control or selects a MenuItem, the 6th element of the event message contains the name of that control or MenuItem. If the user switches to another application, the 6th element of the event message is an empty character vector. In all these cases, the 7th and 8th elements are 0.

The 5th element of the event message contains the data value that will be used to update the Values property. This will be numeric if the FieldType of the associated Edit object is Numeric, LongNumeric, Date, LongDate or Time. Otherwise, it will be a character array.

An application can update an individual cell in the Grid under program control by calling CellChange as a method. If so, the *New object*, *New cell row* and *New cell column* parameters may be omitted.

CellChanged   Event 164

Applies to   Grid

If enabled, this event is reported after the user has changed the contents of a cell in a Grid object and then moved to another cell or to another control outside the Grid. The purpose of this event is to give the application the opportunity to perform calculations, and perhaps to update other cells in the Grid as a result of the change.

Note that this event is reported **after** the change has taken place, and after the Values property has been updated. Furthermore, neither setting the event action code to ¯1 nor returning 0 from a callback function has any effect. If you wish to *validate* the new data you should use the CellChange (150) event instead.

The event message reported as the result of □DQ, or supplied as the right argument to your callback function, is an 5-element vector as follows:

[1] Object:		ref or character vector
[2] Event code:		*'CellChanged'* or 164
[3] Cell row:		integer
[4] Cell column:		integer
[5] New data:		number or character array

The 5th element of the event message contains the data value that has been used to update the Values property. This will be numeric if the FieldType of the associated Edit object is Numeric, LongNumeric, Date, LongDate or Time. Otherwise, it will be a character array.

If you want to update an individual cell under program control, you may call CellChange, but not CellChanged, as a method.

CellDblClick   Event 163

Applies to   Grid

If enabled, this event is reported when the user double-clicks a mouse button whilst over a cell in a Grid. The purpose of this event is to allow an application to enable some special action on double-click. This event may not be disabled.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is an 8 element vector as follows :

[1] Object:		ref or character vector
[2] Event code:		*'CellDblClick'* or 163
[3] Y:		y-position of mouse (number)
[4] X:		x-position of mouse (number)
[5] Button:		button pressed (number)
		1 = left button
		2 = right button
		4 = middle button
[6] Shift State:		sum of shift key codes (number)
		1 = Shift key is down
		2 = Ctrl key is down
		4 = Alt key is down
[7] Cell row:		integer
[8] Cell column:		integer
[9] Title index		integer

The y and x position of the mouse are reported relative to the top-left corner of the first cell in the Grid.

The cell row and column are $\square IO$ dependent.

If the user clicks over a row *title*, the value reported for the column is ¯1, and the value reported for Title index is the index of that row title in RowTitles, or, if RowTitles is not defined, the row number. Column titles are handled in a similar fashion.

CellDown   Event 161

Applies to   Grid

If enabled, this event is reported when the user presses a mouse button down whilst over a cell in a Grid. The purpose of this event is to allow an application to display a pop-up Menu or a Locator over a cell in a Grid or to take some other special action.

The default action is to generate a CellMove event which will then position the user on the new cell. This action can be prevented by returning 0 from the callback function, in which case the normally ensuing CellMove event will not occur.

The event message reported as the result of $\Box DQ$, or supplied as the right argument to your callback function, is a 9 element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	*'CellDown'* or 161
[3] Y:	y-position of mouse (number)
[4] X:	x-position of mouse (number)
[5] Button:	button pressed (number) 1 = left button 2 = right button 4 = middle button
[6] Shift State:	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down 4 = Alt key is down
[7] Cell row:	integer
[8] Cell column:	integer
[9] Title index	integer

The y and x position of the mouse are reported relative to the top-left corner of the first cell in the Grid.

The cell row and column are $\Box IO$ dependent.

If the user clicks over a row *title*, the value reported for the column is ¯1, and the value reported for Title index is the index of that row title in RowTitles, or, if RowTitles is not defined, the row number. Column titles are handled in a similar fashion.

An application **can** position the user on a particular cell in a Grid by calling CellDown as a method, but it is recommended that a CellMove event is used instead.

CellError   Event 157

Applies to   Grid

If enabled, this event is reported when the user inserts invalid data into the Edit object associated with a cell in a Grid object and then attempts to move to another cell or to another control outside the Grid. It is also reported if the user selects a MenuItem.

The default action for the CellError event is to sound the bell (beep). This action can be disabled by returning 0 from the attached callback function. Whatever the result of the callback, the user will be prevented from moving to another cell in the Grid and the CurCell and Values properties will remain unchanged. The user is not prevented from switching to any other control or to another application. However, if and when the user returns to the Grid, the current cell (CurCell) remains the invalid one and the user may not select a different one until the invalid data in the cell has been corrected. If you wish to allow the user to move to another cell without correcting the data, you may do so by generating a CellMove event explicitly. However, the Values property will remain unchanged and the invalid contents of the Edit object will simply be discarded.

The event message reported as the result of $\Box DQ$, or supplied as the right argument to your callback function, is an 8-element vector as follows:

[1] Object:		ref or character vector
[2] Event code:		*'CellError'* or 157
[3] Cell row:		integer
[4] Cell column:		integer
[5] Invalid data:		character vector
[6] Object name:		ref or character vector (object to which the user has transferred focus)
[7] New cell (row):		integer
[8] New cell (column):		integer

If the user moves to another cell in the Grid, the 6th element of the event message is the name of the Grid object and elements 7 and 8 specify the new cell address (`⎕IO` dependent).

If the user switches the input focus to another control or selects a MenuItem, the 6th element of the event message contains the name of that control or MenuItem. If the user switches to another application, the 6th element of the event message is an empty character vector. In all these cases, the 7th and 8th elements are 0.

The 5th element of the event message contains the character vector in the Text property of the associated Edit object which is inconsistent with its FieldType.

CellFonts   Property

Applies to   Grid

This property specifies the name or names of font objects to be used to display the Values in a Grid object.

CellFonts is either a simple character scalar or vector, or a vector of character vectors. If it is simple, it specifies the name of a single font object to be used to draw all of the cells in the Grid. If it is nested, it specifies a set of font objects which are mapped to individual cells through the CellTypes property.

CellFromPoint   Method 200

Applies to   Grid

This method converts from Grid co-ordinates to cell co-ordinates.

The argument to CellFromPoint is a 2-element array as follows:

      [1] y-coordinate: number in Grid co-ordinates

      [2] x-coordinate: number in Grid co-ordinates

The result is a 2-element vector containing the following:

      [1] y-coordinate: number in cell co-ordinates

      [2] x-coordinate: number in cell co-ordinates

CellHeights   Property

Applies to   Grid

This property specifies the height of each row in a Grid object in the units specified by its Coord property. It may be a scalar or a vector whose length is the same as the number of rows implied by the Values property. If it is a scalar, it specifies a constant row height. If it is a vector it specifies the height of each row individually.

CellMove   Event 151

Applies to   Grid

If enabled, this event is reported when the user attempts to position the cursor over a cell in a Grid by clicking the left mouse button or by pressing a cursor movement key. The purpose of this event is to allow an application to perform some action prior to the user entering a cell, or to inhibit entry to a cell.

The default action is to position the user on the new cell. This action can be prevented by returning a 0 from the callback function attached to the event.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 7 element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	*'CellMove'* or 151
[3] New cell row:	integer
[4] New cell column:	integer
[5] Scroll flag:	0 or 1
[6] Selection flag	0, 1 or 2
[7] Mouse flag	0 or 1

The 5th element of the event message is 1 if switching to the new cell caused the Grid to scroll.

The 6th element of the event message is 1 if the user is moving to the new cell by extending the selection. It is 2 if the user selects an entire row or column (by clicking on a title), which moves the current cell to the first one in the selection.

The 7th element of the event message is 1 if the mouse is used to switch to a new cell.

An application can position the user on a particular cell in a Grid by calling CellMove as a method. If so, the argument need contain only the *New cell row* and *New cell column* parameters.

CellSelect   Property

Applies to   Grid

The Grid supports the selection of contiguous and non-contiguous blocks of cells by the user, using the mouse and/or the keyboard. The ability to select a range of cells is determined by the CellSelect property. This may be a character vector or a vector of character vectors comprising the following :

'Rows'	User may select an entire row by clicking on a row title and may select contiguous multiple rows by dragging the mouse over contiguous row titles.
'MultiRows'	Same as *'Rows'*, but user may additionally select several non-contiguous rows and blocks of rows using the Ctrl key.
'Columns'	User may select an entire column by clicking on a column title and may select multiple columns by dragging the mouse over contiguous column titles.
'MultiColumns'	Same as *'Columns'*, but user may additionally select several non-contiguous columns and blocks of columns using the Ctrl key.
'Partial'	User may select any rectangular block of cells by either dragging the mouse or using Shift+cursor keys.
'MultiPartial'	Same as *'Partial'*, but user may additonally select multiple rectangular blocks of cells using the Ctrl key.
'Whole'	User may select the entire Grid by clicking in the space to the left of the column titles and above the row titles.
'Any'	Same as (*'Rows'* *'Columns'* *'Partial'* *'Whole'*). This is the default.
'Multi'	Same as (*'MultiRows'* *'MultiColumns'* *'MultiPartial'* *'Whole'*).
'None'	User may not select any cells in the Grid.

For example, the following expression would allow the user to select only contiguous rows and columns:

> *Gridname.CellSelect←'Rows' 'Columns'*

The following expression would allow the user to select all cells, a contiguous block of cells, or multiple rows and blocks of rows.

> *Gridname.CellSelect←'Whole' 'Partial' 'MultiRows'*

When the user performs a selection , the Grid generates a GridSelect event.

The range of cells currently selected is given by the SelItems property. You can obtain the current selection by querying this property with ⎕WG and you can set it with ⎕WS.

Note that the user may delete the contents of the selected range, or cut and copy them to the clipboard by pressing Delete, Shift+Delete or Ctrl+Insert respectively. The user may also replace the current selection with the contents of the clipboard by pressing Shift+Insert. These operations generate GridDelete, GridCut, GridCopy and GridPaste events which you may disable (by setting the event action code to ¯1 or to which you may attach a callback function.

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

You can also invoke these events as methods. This allows you to attach these actions to MenuItems and Buttons. For example, the following expression could be used to implement Cut as a MenuItem:

```
name ⎕WC 'MenuItem' 'Cu&t'
    ('Event' 'Select' '⍙gridname.GridCut')
```

In addition to the ability to copy blocks of cells through the clipboard, the user may also drag a block of cells from one part of the Grid to another.

If the user has selected a single contiguous block of cells, and then places the mouse pointer over any of the four edges of a selected block of cells, the cursor changes from a cross to an arrow pointer. The user may now drag the border of the selected block to a new location. If the Ctrl key is pressed at the same time, the contents of the selected cells are *copied* to the new location. If not, the operation is a *move* and the original block of cells is cleared (emptied). In either case, the contents of the original block replace the contents of the target block (marked by the dragging rectangle) and the target block become selected.

These operations generate a GridDropSel event. You may prevent the user from moving and copying blocks of cells by disabling this event (by setting its event action code to ¯1) or you may control these operations selectively with a callback function. Note that although the operation of *inserting* cells (using Ctrl+Shift) has not been implemented, you may provide this facility yourself with the information provided by the event message. You may also move or copy a block of cells (which need not necessarily be selected) under program control by calling GridDropSel as a method.

CellSet   Property

Applies to   Grid

This property identifies which cells in a Grid are *set* (i.e. have values) and which are empty. Its purpose is to allow large numeric matrices containing blank cells to be displayed and edited efficiently.

The CellSet property is a Boolean matrix with the same shape as the Values property. If an element of CellSet is 0, the cell is defined to be empty. Empty cells are displayed as blank and the corresponding elements of the Values property are ignored.

A more direct way to handle empty cells is to set the corresponding elements of Values to empty vectors. However, if Values is otherwise entirely numeric, this makes the array nested when it would otherwise be simple. For large numeric matrices, this penalty can be severe. For example, a 100x100 array of 2-byte integer values occupies about 20Kb of workspace. Setting one or more elements of the array to an empty vector increases its size to 200Kb. However, because it is Boolean, the size of the CellSet property for a 100x100 array is only 1.27Kb and represents a significant saving of space.

Note that if the Values property contains text and is therefore nested anyway, the CellSet property is not helpful in conserving workspace, although it may still be useful to separate empty cells from real data.

You can dynamically change a single element of CellSet using the SetCellSet method.

CellTypes   Property

Applies to   Grid

This property specifies the type of each cell in a Grid object. It is a matrix whose elements are indices into other property arrays (including FCol, BCol, CellFonts and Input).

For example, if CellTypes[1;1] is 3, the first cell in the Grid is displayed using the foreground colour specified by the 3rd element of FCol, the background colour specified by the 3rd element of BCol, and so forth. Note however that scalar property arrays are extended if necessary. Therefore if you require 5 different foreground colours but only one background colour, BCol need specify only a single colour.

You can dynamically change a single element of CellTypes using the SetCellType method.

CellUp   Event 162

Applies to   Grid

If enabled, this event is reported when the user releases a mouse button down whilst over a cell in a Grid. This event is a companion to the CellDown event and could be used to hide a pop-up which was displayed in response to the CellDown. The CellUp event performs no default action and may not be disabled.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is an 9 element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	*'CellUp'* or 162
[3] Y:	y-position of mouse (number)
[4] X:	x-position of mouse (number)
[5] Button:	button released (number) 1 = left button 2 = right button 4 = middle button
[6] Shift State:	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down 4 = Alt key is down
[7] Cell row:	integer
[8] Cell column:	integer
[9] Title index	integer

The y and x position of the mouse are reported relative to the top-left corner of the first cell in the Grid.

The cell row and column are $\square IO$ dependent.

If the user clicks over a row *title*, the value reported for the column is ¯1, and the value reported for Title index is the index of that row title in RowTitles, or, if RowTitles is not defined, the row number. Column titles are handled in a similar fashion.

CellWidths Property

Applies to Grid

This property specifies the width of each column in a Grid object in the units specified by its Coord property. It may be a scalar or a vector whose length is the same as the number of columns implied by the Values property. If it is a scalar, it specifies a constant column width. If it is a vector it specifies the width of each column individually.

Change Event 36

Applies to Combo, Edit, RichEdit, Spinner

If enabled, this event is reported when the user alters the text in a Combo or Edit object (by typing). The event is not applicable for a Combo with Style *'Drop'* because this Style does not allow the user to alter data. The Change event is not reported repeatedly as the user edits the data. Instead, it is reported when the user indicates that he has finished with the field by :

a) clicking on another object

or

b) causing an event on another object (without altering the input focus) which will fire a callback function or cause ⎕*DQ* to terminate. This can occur if the user chooses a MenuItem, or fires a Button with the Default or Cancel property by pressing Enter or Esc, or selects an object using an accelerator key.

The purpose of the Change event is to allow the application to validate data which has been newly entered to the field, before proceeding with another action. It is for this reason that the event is fired not just when the input focus changes, but also when the user takes some action that could cause the application to do something else.

The event message reported as the result of ⎕*DQ*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	*'Change'* or 36
[3] Object:	ref or character vector (object that is to receive the focus or generate an event)

If the focus is transferred to an external application, the third element is an empty vector.

The default processing for this event is to allow the focus to change (if applicable) and to reset the internal flag that indicates that the data in the field has changed.

If you disable the event by setting the "action" code to ¯1, or inhibit it by returning 0 from a callback, the focus change (if applicable) is allowed to proceed, but the internal flag is not reset. If you wish to prevent the focus change you must explicitly reset the focus back onto the object that generated the event.

If the event was generated because the user switched to another application, and you return a 0 from your callback (because the data was not valid), the flag marking the Combo or Edit as having been changed remains set. If the user returns to your application by re-focusing on the same Combo or Edit, nothing happens immediately, but because the field is marked as changed (the flag was not reset) you will get another Change event when he leaves it. However, if the user returns to your application in some other way, e.g. by focusing on another object or by selecting a MenuItem, a second Change event will be generated immediately.

The following function illustrates how Change events can be processed. The *Check* function referred to in line[4] is assumed to return 1 if the data is valid and 0 if not.

```
[0]    R←VALIDATE Msg
[1]  ⍝ Validates field contents after Change event
[2]
[3]  ⍝ Normal exit (R←1) if data is valid
[4]    →(R←Check⊃Msg)/Exit
[5]
[6]  ⍝ R now 0, so field remains marked as "changed"
[7]
[8]  ⍝ If user has switched to another application,
[9]  ⍝ we need take no further action because we will
[10] ⍝ get a second Change event when he returns.
[11]   →(''≡3⊃Msg)/Exit
[12]
[13] ⍝ Display error box (prepared earlier)
[14]   'ERR' ⎕WS 'Text' 'Data is invalid'
[15]   ⎕DQ'ERR'
[16]
[17] ⍝ Restore focus to bad field
[18]   ⎕NQ(⊃Msg)40
[19]
[20] Exit:
```

Changed Property

Applies to   Edit, PropertyPage, RichEdit, Spinner

The Changed property, in conjunction with the Change event, provides the means to control the validation of an object after the user has finished interacting with it.

Initially, the value of the Changed property of an object is set to 0. When the user gives the focus to the object and causes either the Text or (in the case of a Spinner) the Thumb property to be altered, the Changed property is immediately set to 1. When the object loses the input focus and the value of the Changed property is 1, the object generates a Change event. The value of the Changed property is then determined as follows:

- If there is no callback function attached to the Change event, or if the Change event is disabled, the Changed property is reset to 0.

- If an attached callback returns no result or returns 1, the Change property is reset to 0.

- If an attached callback function returns 0, the Changed property is not altered and remains set to 1. The object will therefore generate another Change event when the user next tries to leave it, even if the text and/or Thumb are not altered this time.

Note that the object generates a Change event when it loses the focus *only* if the value of the Changed property is 1 at the time.

CharFormat   Property

Applies to   RichEdit

The CharFormat property describes or applies formatting to the currently selected text in a RichEdit object. If the selection is empty, it reports or specifies the default character formatting for the object. It is a 5-element nested array structured as follows:

[1]   A vector of character vectors which describes the text attributes and is comprised of the following keywords:

        `'Autocolour'`   default colour (Windows text colour)
        `'Bold'`   bold text
        `'Italic'`   italic text
        `'Underline'`   underlined
        `'StrikeOut'`   line through text
        `'Protected'`   protected (read-only) text

[2]   A character vector that specifies the face name of the font used to draw the text

[3]   Character height in Twips.

[4]   Text colour. A single integer or an enclosed vector of 3 RGB values. The default is 0 which implies the standard Windows text colour.

[5]   Integer specifying the vertical offset of the character from the base line in Twips. This is used to specify superscript (positive offset) and subscript (negative offset) symbols. The default value is 0.

When you set the character format using `⎕WC` or `⎕WS` the following rules apply.

- If you just want to set a single text attribute (element 1) you may specify a simple vector, for example (`⎕WS 'CharFormat' 'Protected'`) is valid and will add the protected text attribute to the current set of text attributes.

- To cancel a text attribute (element 1) you must insert the tilde (~) character before the name of the attribute. For example, the expression (`⎕WS 'CharFormat' '~Bold'`) will turn the bold text attribute off.

- You need only specify the number of elements required, but you must insert proper values for the elements you wish to remain unaltered. However, you may use `''` in the first element to leave the text attributes unchanged.

If there is no text selected, CharFormat specifies the *default* character format, i.e. the format that will be used to draw the next (and subsequent) characters that the user enters. If there *is* text selected it specifies the format of the selected block of text. If the format is not strictly homogeneous, `⎕WG` may report the format of the first character in the selected block, or, if the block contains characters which use completely different fonts, the result of (`⎕WG 'CharFormat'`) will be empty.

(`⎕WS 'CharFormat' ...`) will set the format of the currently selected block of text.

To set the format of an arbitrary block of text you must select it first using (`⎕WS 'SelText' ...`).

CheckBoxes   Property

Applies to   ListView, TreeView

Requires the Windows Custom Control Library COMCTL32.DLL, Version 4.70 or higher.

The CheckBoxes property specifies whether or not check boxes are displayed alongside items in a ListView or TreeView object.

CheckBoxes is a single number with the value 0 (check boxes are not displayed) or 1 (check boxes are displayed); the default is 0.

Note that check boxes will only be displayed if the object also displays images. You cannot use CheckBoxes without images.

For a ListView, CheckBoxes applies to all settings of the View property.

The GetItemState method can be used to determine if a specific item in a ListView or TreeView is checked. The result of the method will have the 13th bit set if the item is checked.

```
    STATE←Form.ListView.GetItemState 11
    13⊃⌽(32⍴2)⊤STATE
1
```

The SetItemState method may be used to toggle the state of a check box for a particular item.

The picture below illustrates the effect on the appearance of a ListView object, of setting CheckBoxes to 1.
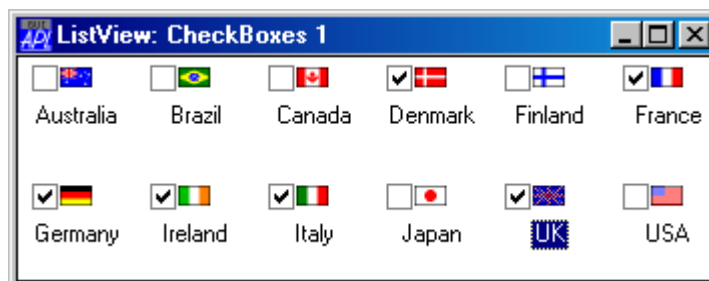
Checked   Property

Applies to   MenuItem

This property determines whether or not a check mark or radio button (according to the value of Style) is displayed alongside the text in a MenuItem.

Checked is a single number with the value 0 (not checked) or 1 (checked). The default is 0.

ChildEdge   Property

Applies to   CoolBand

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

ChildEdge is a single number with the value 0 (no space) or 1 (space is provided); the default is 0.

ChildList   Property

Applies to   ActiveXContainer, ActiveXControl, Animation, Bitmap, BrowseBox, Button, Calendar, Circle, Clipboard, ColorButton, Combo, ComboEx, CoolBand, CoolBar, Cursor, DateTimePicker, Edit, Ellipse, FileBox, Font, Form, Grid, Group, Icon, Image, ImageList, Label, List, ListView, Locator, Marker, MDIClient, Menu, MenuBar, MenuItem, Metafile, MsgBox, OCXClass, OLEClient, OLEServer, Poly, Printer, ProgressBar, PropertyPage, PropertySheet, Rect, RichEdit, Root, Scroll, Separator, SM, Spinner, Splitter, Static, StatusBar, StatusField, SubForm, SysTrayItem, TabBar, TabBtn, TabButton, TabControl, TCPSocket, Text, Timer, TipField, ToolBar, ToolButton, ToolControl, TrackBar, TreeView, UpDown

This is a read-only property that reports the types of those objects that may be created as children of the object in question.

ChildList is a vector of character vectors in which the order of the items is not significant.

ChooseFont   Method 240

Applies to   ActiveXControl, Button, Calendar, Combo, ComboEx,
                      DateTimePicker, Edit, Font, Form, Grid, Group, Label, List,
                      ListView, PropertyPage, PropertySheet, RichEdit, Root, Spinner,
                      Static, StatusBar, SubForm, TabBtn, Text, TipField, TreeView

This method is used to display the standard Windows font selection dialog box.

The argument to ChooseFont is 0 or a 1 or 2-element array as follows:

> [1] Printer name: character scalar or vector.
>
> [2] Modify flag: 0 or 1.

If the argument is 0 or the first element of the argument is ' ', the user is offered a list
of fonts suitable for use on the screen. If not, the user is offered a choice of fonts
suitable for the specified Printer object. If you omit the 2nd element, the modify flag
defaults to 0.

The dialog box is initialised with the properties of the Font object specified in the first
element of the event message.

When the user presses the "OK" button, the "Cancel" button or closes the dialog box,
ChooseFont terminates. Its result is either 0 (user pressed "Cancel") or a 2-element
vector. In the latter case, the first element is a 7-element array that describes the selected
font (see FontObj property) and the second element is a 3-element RGB colour vector.

If the modify flag was 1, the Font object is redefined to match the user's selections and
all the objects that reference the Font are redrawn.

Circle   Object

Purpose	A Graphical object to draw circles, arcs, and pie-slices.
Parents	ActiveXControl, Animation, Bitmap, Button, Combo, ComboEx, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, Metafile, Printer, ProgressBar, PropertyPage, PropertySheet, RichEdit, Scroll, Spinner, Static, StatusBar, SubForm, TabBar, TipField, ToolBar, TrackBar, TreeView, UpDown
Children	Timer
Properties	Type, Points, Radius, FCol, BCol, Start, End, ArcMode, LStyle, LWidth, FStyle, FillCol, Coord, Visible, Event, Dragable, OnTop, CursorObj, AutoConf, Data, Accelerator, KeepOnClose, DrawMode, MethodList, ChildList, EventList, PropList
Events	Close, Create, DragDrop, Help, MouseDblClick, MouseDown, MouseMove, MouseUp, Select
Methods	Detach

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

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

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

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

If Start is specified, but not End, end angles default to (¯1↓+*Start*),○2. If End is specified, but not Start, start angles default to 0,¯1↓+*End*

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

Examples

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

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

An arc

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

Complete pie

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

'g.pie' □WC 'Circle' (50 50) 20 ('Start' ANGLES)
           ('ArcMode' 2) ('FCol' (⊂0 0 0))
           ('FStyle' PATS) ('FillCol' COLS)
```

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

```
CY←50 52 50 50  ⍝ y-coord of centres
R←20 20 20 17.5 ⍝ radii

'g.pie' □WC 'Circle' (50 CY) R ('Start' ANGLES)
           ('ArcMode' 1) ('FCol' (⊂0 0 0))
           ('FStyle' PATS) ('FillCol' COLS)
```

CircleToday   Property

Applies to   Calendar, DateTimePicker

The CircleToday property specifies whether or not a circle is drawn around the Today date in a Calendar object, or in the drop down calendar in a DateTimePicker, when the month containing that date is visible.

CircleToday is a single number with the value 0 (a circle is *not* drawn) or 1 (a circle *is* drawn); the default is 1.

See also HasToday property.

ClassID   Property

Applies to   ActiveXControl, OCXClass, OLEClient, OLEServer

The ClassID property specifies the class identifier (usually abbreviated to CLSID) of an APL object that is used to represent a COM object. The CLSID is a globally unique identifier (GUID) that uniquely identifies the object.

When you create or recreate an ActiveXControl or OLEServer using *⎕WC*, you may specify ClassID. This allows you to re-use a value that was previously allocated to that control by the system. However, you should not specify any other value because that value could be allocated now or in the future to another object *on any other computer in the world*. Otherwise, a new ClassID is automatically allocated by the system.

Note that the CLSID is not actually recorded on your computer (in the registry) until you register it using *)SAVE* or *Make OCX*, or by executing the OLERegister method.

ClassName   Property

Applies to   ActiveXControl, OCXClass, OLEClient, OLEServer

For an OLEClient, the ClassName property specifies the name of the OLE object to which an OLEClient object named by the left argument of □*WC* is to be connected. Note that ClassName is mandatory for □*WC* and may not subsequently be changed using □*WS*.

For an ActiveXControl or OLEServer, ClassName specifies the external name with which the object is registered, and by which it is referenced by other applications.

For an ActiveXControl, the external name is "Dyalog xxx Control, where xxx is the value of the ClassName property, or, if ClassName is not specified, the name of the ActiveXControl namespace.

For an OLEServer, the external name is "Dyalog.xxx" where xxx is derived in the same way.

ClickComment   Event 225

Applies to   Grid

If enabled, a ClickComment event is generated when the user clicks the mouse in a comment widow.

The event message reported as the result of □*DQ*, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	*'ClickComment'* or 225
[3] Row:	integer
[4] Column:	integer

The event message reports the co-ordinates of the cell. The default action is to raise the comment window so that it appears above all other, potentially overlapping, comment windows.

Clipboard   Object

Purpose	This object provides access to the Windows clipboard.
Parents	ActiveXControl, CoolBand, Form, OLEServer, PropertyPage, PropertySheet, Root, TCPSocket
Children	Timer
Properties	Type, Event, Data, Formats, Text, Bits, CMap, CBits, MetafileObj, Picture, Array, RTFText, Translate, Accelerator, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	ClipChange, Close, Create, Select
Methods	Detach, Wait

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

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

Data is read from the clipboard using *□WG*, specifying the appropriate name of the property for the data that you want.

If the data has been stored in CF_TEXT format, the value of Formats will include *'Text'* and you may retrieve the data by querying the value of the Text property with *□WG.*

If the data has been stored in **device-independent** bitmap format, the value of Formats will include *'CBits'*, *'Bits'* and *'CMap'*. To retrieve the bitmap pattern and colour map, you may query the values of the CBits, or Bits and CMap properties (according to the palette size) using *□WG.*

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

The following example retrieves text from the clipboard :

```
'CL' ⎕WC 'Clipboard'
Data ← CL.Text
```

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

```
'BM' ⎕WC 'Bitmap' ('CBits' CL.CBits)
```

Data may be placed in the clipboard using assignment, ⎕WC or ⎕WS. To store text, you simply set the Text property. You may use a simple character vector or matrix, or a vector of character vectors. For example :

```
CL.Text ← 'Hello World'
```

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

```
CL.Picture ← 'BM'
```

or

```
CL.CBits ← BM.CBits
```

Note that if you use Bits and CMap, you must set **both** properties in one ⎕WS statement. This is also true if you wish to store data in more than one format.

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

The Array property allows you to use the Windows clipboard to store and retrieve an arbitrary Dyalog APL array.

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

ClipCells   Property

Applies to   Grid

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

The first picture below shows a default Grid (ClipCells is 1) in which the third column of data is in fact incomplete (clipped), although this is by no means apparent to the user. The second picture shows the effect on the Grid of setting ClipCells to 0 which prevents such potential confusion.

ClipCells is 1 (the default)			
	A	B	C
1	13	75	4
2	51	83	
3	68	58	9
4	76	26	
5	98	72	7

ClipCells is 0		
	A	B
1	13	75
2	51	83
3	68	58
4	76	26
5	98	72

ClipChange   Event 120

Applies to   Clipboard

If enabled, this event is reported when another application changes the contents of the Windows clipboard.

The event message reported as the result of □*DQ*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	*'ClipChange'* or 120

Close   Event 33

Applies to   ActiveXContainer, ActiveXControl, Animation, Bitmap, BrowseBox, Button, Calendar, Circle, Clipboard, ColorButton, Combo, ComboEx, CoolBand, CoolBar, Cursor, DateTimePicker, Edit, Ellipse, FileBox, Font, Form, Grid, Group, Icon, Image, ImageList, Label, List, ListView, Locator, Marker, MDIClient, Menu, MenuBar, MenuItem, Metafile, MsgBox, NetType, OLEServer, Poly, Printer, ProgressBar, PropertyPage, PropertySheet, Rect, RichEdit, Scroll, Separator, SM, Spinner, Splitter, Static, StatusBar, StatusField, SubForm, SysTrayItem, TabBar, TabBtn, TabButton, TabControl, TCPSocket, Text, Timer, TipField, ToolBar, ToolButton, ToolControl, TrackBar, TreeView, UpDown

A Close event is generated when an object is destroyed.

For a Form or SubForm, the event may be generated by the user selecting "Close" from its System Menu. In this case, the event is reported **before** the window is destroyed, and you may prevent it from going ahead by associating a callback function which returns a result of 0.

By trapping this event you can control termination of your application in many different ways. For example, you can :

- automatically close all Forms in your application when the master Form is closed.
- prevent the user from terminating the application if it is inappropriate at that time.

- display an "Are you sure ?" MsgBox.

The event message reported as the result of ⎕*DQ*, or supplied as the right argument to your callback function is a 2-element vector as follows :

> [1] Object: ref or character vector
>
> [2] Event code: *'Close'* or 33

CloseUp   Event 46

Applies to   DateTimePicker

If enabled, this event is reported by a DateTimePicker object just before the drop-down calendar is hidden. It applies only if the Style of the DateTimePicker is *'Combo'*.

The event message reported as the result of ⎕*DQ*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

> [1] Object: ref or character vector
>
> [2] Event name or code: *'CloseUp'* or 46

This event is reported for information only and cannot be disabled or modified in any way.

CMap   Property

Applies to   Bitmap, Clipboard, Cursor, Icon

This property defines the table of colours (the colour map) used by a Bitmap or Icon object or by a bitmap stored in the Windows clipboard. Its value is a 3-column integer matrix of numbers in the range 0-255. Each row represents a separate colour which is indexed (0-origin) by values in the Bits property. The 3 columns refer to the intensities of the red, green and blue components of colour respectively.

Please note that Bits and CMap may **only** be used to represent an image with a colour palette of **256 colours or less**. If the colour palette is larger, the values of Bits and CMap reported by $\square WG$ will be (0 0). For a high-colour image, use CBits instead.

When you create a Bitmap or Icon by specifying Bits and CMap, the actual colours you obtain are not necessarily those that you specified. This is partly due to hardware restrictions and partly due to the way in which Windows manages colours. Firstly, your display adapter and driver limit the number of pure colours that can be displayed at any one time and therefore define a maximum size for the colour map. For example, on a **standard** VGA you are limited to 16 different pure colours (additional ones are provided by **dithering**).

Secondly, Windows reserves a certain number of colours in the colour map for its own use. When an application requests a new colour (i.e. one that is not already installed in the colour map), Windows either assigns it to a spare entry, or allocates the **closest match** if the colour map is full. The value of Bits and CMap after $\square WC$ reflect the actual colours allocated and may bear little resemblance to the values you assigned to these properties initially.

Note that if you are running 16 colours, Windows reserves all 16 entries in the colour map for its own use. This means that on a 16-colour system, you **cannot** use any colours other than the default ones reserved by Windows. In practice, the "standard" 16-colour CMap is shown in the following table.

Bits[ ]	CMap			Colour
0	0	0	0	Black
1	128	0	0	Dark Red
2	0	128	0	Dark Green
3	128	128	0	Olive Green
4	0	0	128	Dark Blue
5	128	0	128	Dark Magenta
6	0	128	128	Dark Cyan
7	128	128	128	Dark Grey
8	192	192	192	Light Grey
9	255	0	0	Red
10	0	255	0	Green
11	255	255	0	Yellow
12	0	0	255	Blue
13	255	0	255	Magenta
14	0	255	255	Cyan
15	255	255	255	White

The default 16-colour CMap

If you are using a 256-colour set-up, the first 9 and the last 7 entries of the 256-colour CMap are the same as the first 9 and last 7 entries of the 16-colour CMap shown above. The intervening entries represent additional colours or are initially unused (0 0 0). New colours that you specify will be allocated to unused entries until the table is full.

ColChange   Method 159

Applies to   Grid

This method is used to change the data in a column of a Grid object.

The argument to ColChange is a 2-element array as follows:

> [1] Column number: integer
>
> [2] Column data: array

Note that the *Column data* must be a scalar or a vector whose length is equal to the number of rows in the Grid. Its elements may be scalar numbers, character vectors or matrices.

Collate   Property

Applies to   Printer

Specifies whether or not multiple copies of printer output are collated.

Collate is a single number with the value 0 or 1. If Collate is 1, multiple copies of output are collated separately. If Collate is 0, copies are uncollated on output.

Collate is ignored unless Copies is >1.

The default value for Collate is derived from the current printer setting and Collate is only effective if the printer supports this capability.

ColLineTypes   Property

Applies to   Grid

This property specifies the appearance of the vertical grid lines in a Grid object.

ColLineTypes is an integer vector, whose length is normally equal to the number of columns in the Grid. Each element in ColLineTypes specifies an index into the GridLineFCol and GridLineWidth properties, thus selecting the colour and width of the vertical grid lines.

For example, if ColLineTypes[1] is 3, the first vertical grid line in the Grid is displayed using the colour specified by the 3rd element of GridLineFCol, and the width specified by the 3rd element of GridLineWidth. Note that ColLineTypes is not $\Box IO$ dependant, and the value 0 is treated the same as the value 1; both selecting the *first* colour and line width specified by GridLineFCol and GridLineWidth respectively.

The default value of ColLineTypes is an empty numeric vector (θ). If so, all vertical grid lines are drawn using the first element of GridLineFCol and GridLineWidth.

A vertical grid line is drawn down the right edge of its associated column. One pixel is drawn *inside* the column of cells; additional pixels (if any) are drawn *between* that column of cells and the next one to its right.

ColorButton   Object

Purpose	Allows the user to choose a colour.
Parents	ActiveXControl, Form, Grid, Group, PropertyPage, SubForm
Children	(None)
Properties	Type, Caption, Posn, Size, CurrentColor, DefaultColors, CustomColors, OtherButton, Coord, Active, Visible, Event, Sizeable, Dragable, CursorObj, AutoConf, Data, Attach, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, Translate, Accelerator, KeepOnClose, ShowDropDown, Redraw, MethodList, ChildList, EventList, PropList
Events	Close, ColorChange, Configure, ContextMenu, Create, DragDrop, DropDown, DropFiles, DropObjects, Expose, GotFocus, Help, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel
Methods	Animate, Detach, GetFocus, GetTextSize, ShowSIP

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

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

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

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

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

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

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

Note that the PocketPC 2002 colour selection dialog box does not provide the facility to select *custom colours*, so this functionality is not available in Pocket APL.

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

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

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

ColorChange Event 430

Applies to ColorButton

If enabled, this event is reported by a ColorButton object when the user chooses a colour from the colour selection drop-down.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	*'ColorChange'* or 430
[3] New Colour:	3-element integer vector

The 3rd element of the event message contains the RGB value for the selected colour.

Note that the event is reported when the user chooses a colour, whether or not the newly selected colour differs from the one that was previously selected.

This event is reported for information only and cannot be disabled or modified in any way.

ColorMode Property

Applies to Printer

Specifies whether or not printing is done in colour.

ColorMode is a single number with the value 0 or 1. If ColorMode is 1, printing is done in colour. If ColorMode is 0, printing is done using black ink only.

This property only applies to colour printers.

ColTitle3D	Property

Applies to   ListView

The ColTitle3D property is a Boolean value that specifies whether or not the column titles in a ListView object are displayed with a 3-dimensional effect. Its default value is 1. A column heading with a 3-dimensional button appearance may be used to imply that the user may click on it to sort by the values in that column.

ColTitle3D is only relevant if View is '*Report*' and Header is 1. Note that this property may only be set by □*WC* and may not subsequently be changed using □*WS*.

ColTitleAlign	Property

Applies to   Grid, ListView

The ColTitleAlign property specifies the alignment of column titles. For a ListView object this is only relevant only when the View property is set to '*Report*'. ColTitleAlign is either a simple character vector, or a vector of character vectors with the same number of elements as ColTitles.

For a Grid, ColTitleAlign may be: '*Top*','*Bottom*','*Left*','*Right*', '*Centre*','*TopLeft*','*TopRight*','*BottomLeft*', or '*BottomRight*'.

For a ListView object, ColTitleAlign may be '*Left*','*Right*' or '*Centre*'. Also, for a ListView the column data itself is aligned likewise. Note however that the *first* column in a ListView is always left-aligned regardless of the setting of ColTitleAlign. This is a Windows restriction.

Note that both spellings '*Centre*' and '*Center*' are accepted.

ColTitleBCol   Property

Applies to   Grid

The ColTitleBCol property specifies the background colour of the column titles in a
Grid object

ColTitleBCol may be a scalar that specifies a single background colour to be used for all
of the column titles, or a vector that specifies the background colour of each of the
column titles individually. An element of ColTitleBCol may be an enclosed 3-element
vector of integer values in the range 0-255 which refer to the red, green and blue
components of the colour respectively, or it may be a scalar that defines a standard
Windows colour element (see BCol for details). Its default value is 0 which obtains the
colour defined for Button Face.

ColTitleDepth   Property

Applies to   Grid
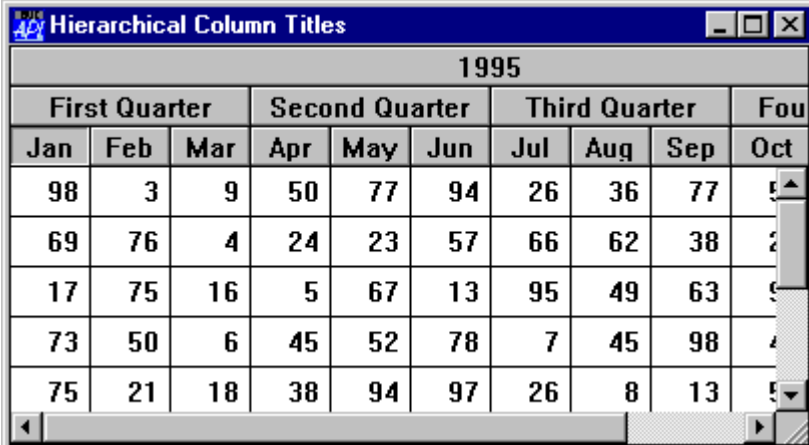
ColTitleDepth specifies the structure of a set of hierarchical column titles. It is an
integer vector with the same length as the ColTitles property. A value of 0 indicates that
the corresponding element of ColTitles is a top-level title. A value of 1 indicates that the
corresponding title is a sub-title of the most recent title whose ColTitleDepth is 0; a
value of 2 indicates that the corresponding title is a sub-title of the most recent title
whose ColTitleDepth is 1, and so forth. For example:

```
'F'⎕WC'Form'('Coord' 'Pixel')('Size' 200 498)
'F'⎕WS'Caption' 'Hierarchical Column Titles'
'F.G'⎕WC'Grid'(?10 12ρ100)(0 0)(200 498)
'F.G'⎕WS('TitleWidth' 0)('TitleHeight' 60)
'F.G'⎕WS'CellWidths' 40

Q1←'First Quarter' 'Jan' 'Feb' 'Mar'
Q2←'Second Quarter' 'Apr' 'May' 'Jun'
Q3←'Third Quarter' 'Jul' 'Aug' 'Sep'
Q4←'Fourth Quarter' 'Oct' 'Nov' 'Dec'

CT←(⊂'1995'),Q1,Q2,Q3,Q4
CD←0,16ρ1 2 2 2

'F.G'⎕WS('ColTitles'CT)('ColTitleDepth'CD)
```

ColTitleFCol   Property

Applies to   Grid

The ColTitleFCol property specifies the colour of the column titles in a Grid object

ColTitleFCol may be a scalar that specifies a single colour to be used for all of the column titles, or a vector that specifies the colour of each of the column titles individually. An element of ColTitleFCol may be an enclosed 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or it may be a scalar that defines a standard Windows colour element (see BCol for details). Its default value is 0 which obtains the colour defined for Button text.

ColTitles   Property

Applies to   Grid, ListView

This property specifies the headings that are displayed above the columns in a Grid or ListView object. If specified, it must be a vector of character vectors or matrices (Grid only).

The default value of ColTitles in a Grid is an empty character vector. In this case, the system displays "standard" spreadsheet column titles A-Z, AA-AZ, BA-BZ and so forth.

To disable the display of column titles in a Grid, you should set the TitleHeight property to 0.

ColumnClick   Event 320

Applies to   ListView

If enabled, this event is reported when the user clicks on the column heading in a ListView object. This event may not be disabled or affected by a callback function in any way.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1] Object:		ref or character vector
[2] Event name or code:		*'ColumnClick'* or 320
[3] Column number:		Integer.
[4] Button:		button pressed (number)
		1 = left button
		2 = right button
		4 = middle button
[5] Shift State:		sum of shift key codes (number)
		1 = Shift key is down
		2 = Ctrl key is down
		4 = Alt key is down

ColumnWidth   Property

Applies to   List

This property specifies the column width in pixels of a multi-column List object. See MultiColumn property for details.

Combo   Object

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

Parents   ActiveXControl, CoolBand, Form, Grid, Group, PropertyPage, SubForm, ToolBar, ToolControl

Children   Circle, Cursor, Ellipse, Font, Marker, Poly, Rect, Text, Timer

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

Events   Change, Close, Configure, ContextMenu, Create, DragDrop, DropDown, DropFiles, DropObjects, Expose, FontCancel, FontOK, GotFocus, Help, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Select

Methods   Animate, ChooseFont, Detach, GetFocus, GetTextSize, ShowSIP

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

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

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

If the Style is `'Simple'` or `'DropEdit'`, the user may type text into the edit field. In these cases, the contents of the edit field may also be specified or queried using the Text property. Note that if the user first selects an item from the list box, then changes it in the edit field, the entry in the list box is automatically deselected. There is therefore no conflict between the value of Text and the value of SelItems.

SIMPLE COMBOS

Chateau Baleau

Chateau Ausone
Clos de l'Angelus
Chateau Baleau
Chateau Balestard-La-Tonne

Chateau Baleau

Chateau Ausone
Clos de l'Angelus
Chateau Baleau
Chateau Balestard-La-Ton
Chateau Beau-Mazerat

DROPEDIT COMBO

Chateau Baleau

Chateau Baleau

Chateau Ausone
Clos de l'Angelus
Chateau Baleau
Chateau Balestard-La-Tonn
Chateau Beau-Mazerat

Normal Appearance   With Listbox Dropped

DROP COMBO

Chateau Baleau

Chateau Baleau

Chateau Ausone
Clos de l'Angelus
Chateau Baleau
Chateau Balestard-La-Tonnel
Chateau Beau-Mazerat

Normal Appearance   With Listbox Dropped

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

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

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

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

Note that when you change the Items property using ⎕*WS*, the Text, SelItems and SelText properties are all reset to their default values.

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

ComboEx   Object

Purpose	The ComboEx object is an extended version of the Combo object that provides additional features including item images
Parents	ActiveXControl, CoolBand, Form, Group, PropertyPage, SubForm, ToolBar, ToolControl
Children	Circle, Cursor, Ellipse, Font, Marker, Poly, Rect, Text, Timer
Properties	Type, Items, Text, Posn, Size, Style, Coord, Rows, Border, Active, Visible, Event, Indents, ImageListObj, ImageIndex, SelImageIndex, CaseSensitive, EditImage, EditImageIndent, PathWordBreak, VScroll, HScroll, SelItems, SelText, Sizeable, Dragable, FontObj, FCol, BCol, CursorObj, AutoConf, Index, Data, Attach, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, Translate, Accelerator, AcceptFiles, KeepOnClose, Redraw, MethodList, ChildList, EventList, PropList
Events	Close, Configure, ContextMenu, Create, DragDrop, DropDown, DropFiles, DropObjects, Expose, FontCancel, FontOK, GotFocus, Help, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Select
Methods	Animate, ChooseFont, Detach, GetFocus, GetTextSize, ShowSIP

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

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

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

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

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

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

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

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

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

Configure   Event 31

Applies to   ActiveXControl, Animation, Button, Calendar, ColorButton, Combo, ComboEx, CoolBar, DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, ProgressBar, PropertyPage, RichEdit, Scroll, SM, Spinner, Static, StatusBar, SubForm, TabBar, ToolBar, ToolControl, TrackBar, TreeView, UpDown

If enabled, this event is generated when the configuration of an object (position and/or size) is about to change.

For a Form, the event is generated when the Form is resized or moved by the user.

For any object other than a Form, it can occur in one of two ways. Firstly, whenever a Form is resized, the system (by default) re-arranges its children so as to maintain their relative position and size. This generates a Configure event (if enabled) for each one of them.

Secondly, it can occur as a result of the user resizing the object directly. This facility is enabled by setting the object's Sizeable property to 1.

Note that a Configure event is not reported when an object is moved using "drag & drop". See Dragable (property) and DragDrop (event) for details of this operation.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	*'Configure'* or 31
[3] Y:	y-position of top left corner
[4] X:	x-position of top left corner
[5] H:	height of object
[6] W:	width of object

For any object, the operation can be prevented by returning a scalar 0 from the callback function associated with the Configure event.

Full-Drag Considerations

Under Windows 95, 98 and NT, the user may choose a system option, described here as *full-drag*, whereby the contents of the window are re-arranged during a resize operation.

If you manage the geometry of your controls using the Attach property, APL honours *full drag* during resize, changing the size and position of your controls dynamically for you.

However, if you manage the geometry of controls using Configure event callbacks, you should consider the following.

If full drag is in enabled, APL generates Configure events *during* the resize operation, allowing you to dynamically alter the geometry of controls as you wish. However, the following restrictions apply:

1. Configure callbacks will only be executed when the interpreter is *idle*. For example during a `⎕DQ` or during Session input (6-space prompt). If the user attempts to move/resize a window that has Configure callbacks attached when the interpreter is busy, the move/resize is not started. This is similar to the operation in non *full drag* mode, where the move/resize is allowed but the callback does not execute until the interpreter again becomes idle.

2. The callback cannot be traced. It is necessary to debug the callback code with *full drag* disabled.

3. Any untrapped errors in the Configure callback will not halt execution in the normal way, but will instead be reported in the *Status Window*. Note that it is also not possible to trap such errors higher up the SI stack than the Configure Callback.

4. There are some programming styles to be avoided if *full drag* Configure callbacks are to be processed correctly. For example events generated by monadic `⎕NQ` within a Configure callback will not be processed until the entire resize operation has been completed.

5. It is not possible to save a workspace from within a Configure Callback in *full drag* mode.

The above restrictions apply to Configure events when *full drag* is enabled, but *only* when *full drag* is enabled. The behaviour of Configure callbacks with *full drag* disabled is the same as for other events.

Container   Property

Applies to   ActiveXControl

The Container property is a read-only property whose value is the `⎕OR` of an ActiveXContainer object that represents the *ActiveX Site* object of the application that is hosting the ActiveXControl.

The value of Container may be converted to a namespace using `⎕NS` or `⎕WC`.

The resulting object may then be used to obtain the values of ambient properties, or to access methods exposed by the host application via OLE interfaces. For further information, see OLEQueryInterface.

ContextMenu   Event 410

Applies to   ActiveXControl, Animation, Button, Calendar, ColorButton, Combo, ComboEx, CoolBar, DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, ProgressBar, PropertyPage, RichEdit, Scroll, Spinner, Static, StatusBar, SubForm, TabBar, ToolBar, ToolControl, TrackBar, TreeView, UpDown

If enabled, this event is reported when the user performs the standard Windows action to display a ContextMenu. These include clicking/releasing the right mouse button and pressing F10.

If the object has its own standard context menu, for example an Edit object, the default action is to display this menu. If the object is dockable (see Docking Property), the default action is to display the standard (English) Dyalog APL docking menu.

You may use this event to display your own pop-up context menu, by `⎕DQ`'ing it within a callback function. In this case, your callback function should return 0 to disable the standard context menu.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	`'ContextMenu'` or 410
[3] Y:	y-position of the mouse (number)
[4] X:	x-position of the mouse (number)

CoolBand   Object

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

Parents   CoolBar

Children   Bitmap, BrowseBox, Button, Clipboard, Combo, ComboEx, Cursor, Edit, FileBox, Font, Grid, Group, Icon, ImageList, Label, List, ListView, Menu, Metafile, MsgBox, OCXClass, OLEClient, Printer, ProgressBar, RichEdit, Scroll, Spinner, Static, StatusBar, SubForm, TabControl, TCPSocket, Timer, TipField, ToolBar, ToolControl, TrackBar, TreeView, UpDown

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

Events   Close, Create, DockAccept, DockCancel, DockEnd, DockMove, DockRequest, DockStart

Methods   Detach

For full functionality as described here, the CoolBand object requires the Windows Custom Control Library COMCTL32.DLL, Version 4.71 or higher.

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

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

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

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

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

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

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

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

The position of a Cool Band within a CoolBar is determined by its Index and NewLine properties, and by the position and size of preceding CoolBand objects in the same CoolBar. For a CoolBand, Posn is a read-only property.

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

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

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

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

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

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

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

CoolBar Object

Purpose	The CoolBar object acts as a container for CoolBand objects.
Parents	ActiveXControl, Form
Children	CoolBand, ImageList, Menu, Timer
Properties	Type, Posn, Size, Align, Event, ImageListObj, FCol, BCol, CursorObj, Data, Attach, Handle, KeepOnClose, BandBorders, DblClickToggle, FixedOrder, VariableHeight, DockChildren, Redraw, MethodList, ChildList, EventList, PropList
Events	Close, Configure, ContextMenu, Create, DockAccept, DockCancel, DockEnd, DockMove, DockRequest, DockStart, DragDrop, DropFiles, DropObjects, Expose, Help
Methods	Animate, Detach, GetFocus, GetTextSize, ShowSIP

The CoolBar object requires the Windows Custom Control Library COMCTL32.DLL, Version 4.71 or higher.

A CoolBar contains one or more bands (CoolBands). Each band can have any combination of a gripper bar, a bitmap, a text label, and a single child object. Using the gripper bars, the user may drag bands from one row to another, resize bands in the same row, and maximise or minimise bands in a row.

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

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

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

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

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

Coord   Property

Applies to   ActiveXControl, Animation, Bitmap, Button, Calendar, Circle,
ColorButton, Combo, ComboEx, DateTimePicker, Edit, Ellipse,
Form, Grid, Group, Image, Label, List, ListView, Locator, Marker,
MDIClient, Menu, Metafile, Poly, Printer, ProgressBar, PropertyPage,
PropertySheet, Rect, RichEdit, Root, Scroll, SM, Spinner, Splitter,
Static, StatusBar, StatusField, SubForm, TabBar, Text, ToolBar,
TrackBar, TreeView, UpDown

This property defines an object's co-ordinate system. It is a character string with one of the following values; `'Inherit'`, `'Prop'`, `'Pixel'`, `'User'` or `'Cell'` (Grid object and graphics children of a Grid only).

If Coord is `'Inherit'`, the co-ordinate system for the object is **inherited** from its parent. Note that the default value of Coord for the system object `'.'` is `'Prop'`, so by default all objects created by `⎕WC` inherit `'Prop'`.

If Coord is `'Prop'`, the origin of the object's parent is deemed to be at its top left interior corner, and the scale along its x- and y-axes is 100. The object's position and size (Posn and Size properties) are therefore specified and reported as a percentage of the dimensions of the parent object, or, for a Form, of the screen.

```
     0              -- X -→        100
  0  +─────────────────────────────────┐
     │                                 │
     │                                 │
  |  │                                 │
  Y  │         Coord 'Prop'            │
  ↓  │                                 │
     │                                 │
 100 └─────────────────────────────────┘
```

If Coord is `'Pixel'`, the origin of the object's parent is deemed to be at its top left interior corner, and the scale along its x- and y-axes is measured in physical pixel units. The object's size and position (Posn and Size properties) are therefore reported and set in physical pixel units. If you set Coord on the system object to `'Pixel'`, the value of its Size property gives you the resolution of your screen, e.g. (480,640). Note that pixels are numbered from 0 to (Size-1).

```
  0                -- X -→
0  ┌─────────────────────────────┐
   │                             │
│  │                             │
Y  │        Coord 'Pixel'        │
↓  │                             │
   │                             │
   │                             │
   └─────────────────────────────┘
```

If Coord is `'User'`, the origin and scale of the co-ordinate system are defined by the values of the YRange and XRange properties **of the parent object**. Each of these is a 2-element numeric vector whose elements define the co-ordinates of top left and bottom right interior corners of the (parent) object respectively. The following examples illustrate the principle.

```
100 ┌─────────────────────────────┐
    │                             │
↑   │                             │
Y   │     YRange 100  0           │
│   │     XRange 0  100           │
    │                             │
  0 └─────────────────────────────┘
    0          -- X -→        100
```

```
 32 ┌─────────────────────────────┐
    │                             │
↑   │                             │
Y   │     YRange 32  ‾2           │
│   │     XRange ‾50  50          │
    │                             │
‾2  └─────────────────────────────┘
   ‾50          -- X -→         50
```
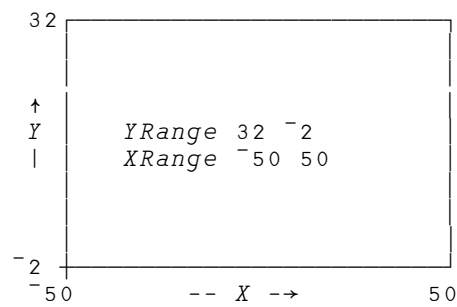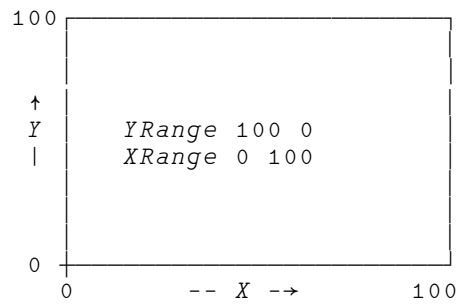
Note that if Coord is `'User'` and you change the values of YRange and/or XRange of the parent, the object (and all its siblings with Coord `'User'`) are redrawn (and clipped) according to the new origin and scale defined for the parent. The values of their Posn, Size and Points properties are unaffected. Changing YRange and/or XRange therefore provides a convenient and efficient means to "pan and zoom".

The Coord property for graphic objects created as a children of a Grid may also be set to Cell. Apart from being easier to compute, a graphic drawn using cell co-ordinates will expand and contract when the grid rows and columns are resized.

Example

This statement creates a button 10 pixels high, 20 pixels wide, and 5 pixels down and along from the top-left corner of the parent Form.

```
'TEST.B1'⎕WC'Button' 'OK'(5 5)(10 20)('Coord' 'Pixel')
```

If you set Coord to `'Pixel'` in the Root object `'.'`, then query its Size, you get the dimensions of the screen in pixels, i.e.

```
      '.' ⎕WS 'Coord' 'Pixel'
      '.' ⎕WG 'Size'
480 640
```

`Copies`   Property

Applies to   Printer

Specifies the number of copies to be printed.

Copies is a non-zero scalar integer value whose default is defined by the current printer settings.

`Create`   Event 34

Applies to   ActiveXContainer, ActiveXControl, Animation, Bitmap, BrowseBox, Button, Calendar, Circle, Clipboard, ColorButton, Combo, ComboEx, CoolBand, CoolBar, Cursor, DateTimePicker, Edit, Ellipse, FileBox, Font, Form, Grid, Group, Icon, Image, ImageList, Label, List, ListView, Locator, Marker, MDIClient, Menu, MenuBar, MenuItem, Metafile, MsgBox, NetType, OLEServer, Poly, Printer, ProgressBar, PropertyPage, PropertySheet, Rect, RichEdit, Scroll, Separator, SM, Spinner, Splitter, Static, StatusBar, StatusField, SubForm, SysTrayItem, TabBar, TabBtn, TabButton, TabControl, TCPSocket, Text, Timer, TipField, ToolBar, ToolButton, ToolControl, TrackBar, TreeView, UpDown

If enabled, this event is reported *after* an object has been created. You may not nullify or modify the event with a 0-returning callback, nor may you generate the event using $\square NQ$, or call it as a method.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	`'Create'` or 34
[3] Flag:	1 = object was created by $\square WC$ 0 = object was created by `)LOAD`, `)COPY` or $\square WC$ of its $\square OR$ representation

This event also applies to the Session object $\square SE$ and may be used to fire a start-up function (in the $\square SE$ namespace) when APL initialises.

CurCell   Property

Applies to   Grid

This property specifies or reports the row and column co-ordinates of the current cell in a Grid object. The current cell is the one that is currently addressed by the user. It is a 2-element integer vector.

CurrentColor   Property

Applies to   ColorButton

The CurrentColor property is a 3-element integer vector that specifies and reports the RGB value of the currently selected colour in a ColorButton object. Its default value is (0 0 0) which is black.

CurrentState   Property

Applies to   TCPSocket

The CurrentState property is a read-only property that reports the current state of a TCPSocket object. Its possible values and their meanings are as follows:

CurrentState	Description
`'Open'`	a client socket that is not yet connected or a UDP socket
`'Bound'`	a server socket that has been bound
`'Listening'`	a server socket to which a client has not yet connected
`'Connected'`	a client or server socket that is connected
`'IhaveClosed'`	a temporary state on the way to Closed
`'PartnerHasClosed'`	a temporary state on the way to Closed
`'Closed'`	a socket that has been closed by both client and server

Cursor   Object

Purpose	This object defines a cursor.
Parents	ActiveXControl, Animation, Button, Calendar, Combo, ComboEx, CoolBand, DateTimePicker, Edit, Form, Grid, Group, ImageList, Label, List, ListView, OLEServer, ProgressBar, PropertyPage, PropertySheet, RichEdit, Root, Scroll, SM, Static, StatusBar, SubForm, TCPSocket, ToolBar, ToolControl, TrackBar, TreeView, UpDown
Children	Timer
Properties	Type, File, Bits, CMap, Mask, HotSpot, KeepBits, Event, Data, Handle, Accelerator, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, Select
Methods	Detach, FileRead, FileWrite

The File property defines the name of a cursor file associated with the Cursor object, or it specifies the name of a DLL and the resource number or name of the cursor therein. Unless specified explicitly, the file extension .CUR is assumed. If supported by the Operating System, you may also specify an animated cursor (.ANI) file. A Cursor is **used** by setting the CursorObj property of another object to its name. If the value of the File property is set by $\Box WS$, no immediate action is taken, but the corresponding file may subsequently be read or written using the FileRead or FileWrite methods.

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

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

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

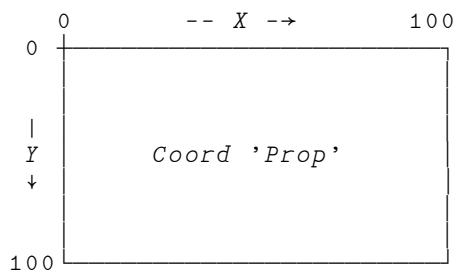# *CursorObj*   Property

Applies to   ActiveXControl, Button, Calendar, Circle, ColorButton, Combo, ComboEx, CoolBar, DateTimePicker, Edit, Ellipse, Form, Grid, Group, Label, List, ListView, Locator, MDIClient, Poly, ProgressBar, Rect, RichEdit, Root, Scroll, SM, Spinner, Splitter, Static, StatusBar, SubForm, TabBar, Text, ToolBar, TrackBar, TreeView, UpDown

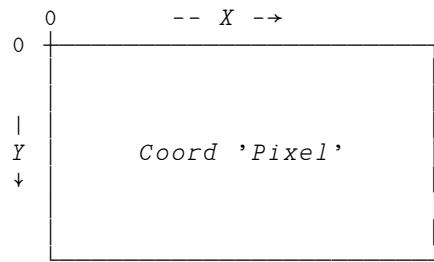This property is used to associate a particular cursor with an object. Its value is either a simple scalar number which specifies a standard Windows cursor, or a character scalar or vector containing the name of a Cursor object. The standard Windows 95 cursors are:

0	arrow (Windows default)
1	hourglass
2	crosshair
3	I-Beam
4	crossing vertical/horizontal double-headed arrows
5	diagonal double-headed arrows (left-to-right)
6	vertical double-headed arrows
7	diagonal double-headed arrows (right-to-left)
8	horizontal double-headed arrows
9	upward pointing arrow
10	box
11	crossing vertical/horizontal double-headed arrows
12	no-entry sign
13	arrow with hourglass

If CursorObj is set to anything other than an empty vector (which is the default) it defines the appearance of the cursor when the mouse pointer is moved into the object. If CursorObj is an empty vector, the shape of the cursor remains unchanged when the mouse pointer enters the object. In effect, the cursor is "**inherited**" from its parent. Exceptions to this rule are certain objects which have special cursors by default.

If the value of CursorObj for the Root object is set to anything other than an empty vector, it applies to **all** Forms and their children, irrespective of their own CursorObj values. Therefore, if you want to indicate that your application is "working" and is not responsive to input, you can simply do :

```
'.' ⎕WS 'CursorObj' 1 ⍝ Hourglass cursor
```

Then to reset the application you do :

```
'.' ⎕WS 'CursorObj' ''
```

CustomColors   Property

Applies to   ColorButton

The CustomColors property is a 1-row, 16-column nested matrix which specifies the RGB values of the colours displayed in the *Custom colors* section of the Windows colour selection dialog box when displayed by a ColorButton object.

Each element of CustomColors is a 3-element integer vector specifying an RGB colour value.

By default, each element of CustomColors is (0 0 0). If the user selects a new custom colour from the Windows colour selection dialog box, its value will be reported by CustomColors. CustomColors must always have shape (1 16).

Note that CustomColors is maintained separately for each separate ColorButton, and CustomColors defaults to (1 16⍴⊂0 0 0) for each new ColorButton that you create. If you want to maintain a global custom colour table for your application, you must do this yourself.

Note that the Pocket PC 2002 colour selection dialog box does not provide the facility to select custom colours, so this functionality is not available in Pocket APL.

CustomFormat   Property

Applies to   DateTimePicker

Specifies a custom format for the date/time display in a DateTimePicker.

CustomFormat is a character vector that may contain a mixture of date/time format elements and body text. The date/time elements are replaced by the actual date/time values when the object is displayed. The body text is displayed *as-is*. Note that CustomFormat may only be specified when the DateTimePicker object is created.

The date/time elements are defined by the following groups of characters, notice that they are case-sensitive:

Element	Description
d	The one- or two-digit day.
dd	The two-digit day. Single-digit day values are preceded by a zero.
ddd	The three-character weekday abbreviation.
dddd	The full weekday name.
h	The one- or two-digit hour in 12-hour format.
hh	The two-digit hour in 12-hour format. Single-digit values are preceded by a zero.
H	The one- or two-digit hour in 24-hour format.
HH	The two-digit hour in 24-hour format. Single-digit values are preceded by a zero.
m	The one- or two-digit minute.
mm	The two-digit minute. Single-digit values are preceded by a zero.
M	The one- or two-digit month number.
MM	The two-digit month number. Single-digit values are preceded by a zero.
MMM	The three-character month abbreviation.
MMMM	The full month name.
t	The one-letter AM/PM abbreviation (that is, AM is displayed as "A").
tt	The two-letter AM/PM abbreviation (that is, AM is displayed as "AM").
yy	The last two digits of the year (that is, 1996 would be displayed as "96").
yyyy	The full year (that is, 1996 would be displayed as "1996").

The body text is defined by sub-strings contained within single quotes. For example, to display the current date with the format "Today is: 04:22:31 Tuesday Mar 23, 1996", the format string is defined as follows:

```
    CustomFormat
'Today is: 'hh':'m':'s dddd MMM dd', 'yyyy
```

To include a single quote in your body text, use two consecutive single quotes. For example, to produce output that looks like: "Don't forget Mar 23, 1996", CustomFormat should be specified as follows:

```
    CustomFormat
'Don''t forget' MMM dd',' yyyy
```

Note non format characters that are not delimited by single quotes will result in unpredictable display by the DateTimePicker object.

Data   Property

Applies to	ActiveXContainer, ActiveXControl, Animation, Bitmap, BrowseBox, Button, Calendar, Circle, Clipboard, ColorButton, Combo, ComboEx, CoolBand, CoolBar, Cursor, DateTimePicker, Edit, Ellipse, FileBox, Font, Form, Grid, Group, Icon, Image, ImageList, Label, List, ListView, Locator, Marker, MDIClient, Menu, MenuBar, MenuItem, Metafile, MsgBox, OCXClass, OLEClient, OLEServer, Poly, Printer, ProgressBar, PropertyPage, PropertySheet, Rect, RichEdit, Root, Scroll, Separator, SM, Spinner, Splitter, Static, StatusBar, StatusField, SubForm, SysTrayItem, TabBar, TabBtn, TabButton, TabControl, TCPSocket, Text, Timer, TipField, ToolBar, ToolButton, ToolControl, TrackBar, TreeView, UpDown

This property allows you to associate arbitrary data with an object. The value of the Data property may be any APL array.

DateTime   Property

Applies to   DateTimePicker

Specifies the value of date/time in a DateTimePicker.

The DateTime property represents the date and time value that is currently displayed in a DateTimePicker object.

It is normally a 4-element integer vector containing the date (as an IDN), hour, minutes and seconds respectively.

However, if the checkbox shown in the object is unset (see HasCheckBox), the value of DateTime will be θ (zilde).

DateTimeChange   Event 267

Applies to   DateTimePicker

If enabled, this event is reported by a DateTimePicker object when the user changes the DateTime value. This occurs when the user selects a new date from the drop-down calendar, or increments or decrements a date time element using the spinner buttons, or edits a datetime element using the keyboard. In the latter case, the event may not be generated until the input focus leaves the corresponding date time element.

The event message reported as the result of *☐DQ*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	*'DateTimeChange'* or 267
[3] IDN	integer
[4] Hour	integer
[5] Minute	integer
[6] Second	integer

This event is reported for information only and cannot be disabled or modified in any way.

DateTimePicker   Object

Purpose	The DateTimePicker object is an editable date/time field with an optional drop-down Calendar.
Parents	ActiveXControl, Form, Group, PropertyPage, SubForm, ToolBar
Children	Cursor, Font, Menu, MsgBox, TCPSocket, Timer
Properties	Type, Posn, Size, Style, Coord, Align, Border, Active, Visible, Event, DateTime, MinDate, MaxDate, CalendarCols, Today, HasToday, CircleToday, WeekNumbers, MonthDelta, HasCheckBox, FieldType, CustomFormat, Sizeable, Dragable, FontObj, CursorObj, AutoConf, Data, Attach, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, Translate, Accelerator, AcceptFiles, KeepOnClose, Redraw, MethodList, ChildList, EventList, PropList
Events	Close, CloseUp, Configure, ContextMenu, Create, DateTimeChange, DragDrop, DropDown, DropFiles, DropObjects, Expose, FontCancel, FontOK, GotFocus, Help, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Select
Methods	Animate, ChooseFont, DateToIDN, Detach, GetFocus, GetTextSize, IDNToDate, ShowSIP

The DateTimePicker object represents the built-in Windows date and time picker control and requires COMCTL32.DLL Version 4.70 or later. For most purposes, the DateTimePicker supersedes the use of Label, Edit and Spinner objects for displaying and entering dates and times. Unlike the Edit and Spinner objects, it is not possible for the user to enter an invalid date or time into a DateTimePicker.

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

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

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

Note that the Style property may only be set when the object is created.

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

The FieldType property specifies one of a set of pre-defined date/time formats to be used by the control. This is a character vector that may be empty (the default), `'Date'`, `'DateCentury'`, `'LongDate'`, `'Time'` or `'Custom'`. Specifying an empty vector is the same as specifying `'Date'`. Note that `'DateCentury'`, which always displays a 4-digit year, regardless of the user's Windows settings, is available only with COMCTL32.DLL Version 5.0 or later.

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

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

DateToIDN   Method 264

Applies to   Calendar, DateTimePicker, Root

This method is used to convert a date from $\square TS$ format into an IDN suitable for use in a Calendar object.

The argument to DateToIDN is a 3-element array as follows:

[1] Year		Integer
[2] Month		Integer
[3] Day		Integer

DateToIDN will also accept a single enclosed argument containing these values. In either case, if you specify more than 3 numbers, excess elements they will be ignored.

Examples

```
      F.C.DateToIDN 1998 9 11
36048
      F.C.DateToIDN ⊂1998 9 11
36048
      F.C.DateToIDN □TS
36048
      F.C.DateToIDN,□TS
36048
```

DblClickToggle Property

Applies to CoolBar

The DblClickToggle property specifies whether or not the user must single-click or double-click to toggle a child CoolBand between its maximised or minimised state.

DblClickToggle is a single number with the value 0 (single-click toggles state) or 1 (double-click toggles state); the default is 0.

DDE Event 50

Applies to Root

If enabled, a DDE event is generated whenever a DDE message is received by Dyalog APL. This will occur whenever a server notifies APL that the value of a shared variable has changed, and whenever a client application requests data from APL. If you have several shared variables, you can determine which of them has changed or whose value has been requested using $\square SVS$.

This event **only** applies to the Root object `'.'`, so to enable it you must execute one of the following statements :

```
        '.' □WS 'Event' 50 1
or
        '.' □WS 'Event' 50 fn
or
        '.' □WS 'Event' 50 fn larg
```

The first statement would cause $\square DQ$ to terminate on receipt of a DDE event. The second would cause it to call *fn* each time. The third would do likewise but the value in *larg* would be supplied as its left argument. Note that due to the nature of DDE *conversations*, messages may be received when in fact no change in the value of any shared variables has occurred. Your application code must therefore be prepared to cater for this situation. The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function is a 2-element vector as follows :

[1] Object name: `'.'` (1-element character vector)

[2] Event code: `'DDE'` or 50

Decimals   Property

Applies to   Edit, Label, Spinner

This property specifies the number of decimal places to which a number is to be displayed in an Edit or Label object with FieldType `'Numeric'`. For an Edit object, Decimals also specifies the maximum number of digits that the user may enter after a decimal point. The default value of decimals is ‾1 which allows any number of decimal places to be entered.

Default   Property

Applies to   Button, MsgBox

This property determines which of a set of push buttons in a Form, SubForm or MsgBox is the default button.

In a Form or SubForm, the Default Button will generate a Select event (30) when the user presses the Enter key, even though the Default Button may not have the focus at the time.

If however, the user explicitly shifts the focus to another Push Button, the automatic selection of the Default Button is disabled and the Enter key applies to the Button with the focus.

For a Button, the Default property has the value 1 or 0. As only one Button can be the Default Button, setting Default to 1 for a particular Button automatically sets Default to 0 for all others with the same parent.

In a MsgBox, Default specifies which button initially has the focus. It has the value 1, 2 or 3 corresponding to the three buttons that can be defined. See Btns property for further details.

DefaultColors   Property

Applies to   ColorButton

The DefaultColors property is a nested matrix which specifies the RGB values of the colours shown in the colour selection drop-down displayed by a ColorButton object.

The shape of DefaultColors determines the number of rows and columns in the colour selection drop-down.

Each element of DefaultColors is a 3-element integer vector specifying an RGB colour value.

DelCol   Method 155

Applies to   Grid

This method is used to delete a specified column from a Grid object.

The argument to DelCol is a 1 or 2-element vector as follows:

[1] Column number:	number of the column (integer) to delete
[2] Undo flag:	0 or 1 (optional; default 0)

If the *Undo flag* 1, the column may subsequently be restored by invoking the Undo method. If the *Undo flag* is omitted or is 0, the operation may not be undone.

DelComment   Method 221

Applies to   Grid

This method is used to delete a comment.

The argument to DelComment is a 2 array as follows or ⊖:

      [1] Row: integer

      [2] Column: integer

For example, the following expression removes the comment associated with the cell at row 2, column 1.

```
F.G.DelComment 2 1
```

If the argument is ⊖, all comments are deleted.

DeleteChildren   Method 311

Applies to   TreeView

This method is used to delete child items from a parent item in a TreeView object.

The argument to DeleteChildren is a scalar or 1 element array as follows:

      [1] Item number: Integer.

Item number specifies the index of the parent item from which the child items are to be removed.

The result is an integer that indicates the number of children that have been removed from the parent item.

DeleteItems   Method 309

Applies to   TreeView

This method is used to delete items from a TreeView object.

The argument to DeleteItems is a 2-element array as follows:

         [1] Item number: Integer.

         [2] Number of Items: Integer.

Item number specifies the index of the first item to be removed.

Number of items specifies the number of items to be removed and refers to those items *at the same level* in the TreeView hierarchy as the *Item number*. *Number of items* is optional and defaults to 1.

Note that any children of these items will also be removed.

The result is an integer that indicates the total number of items, including children, that have been removed from the TreeView.

DeleteTypeLib   Method 521

Applies to   Root

The DeleteTypeLib method removes a loaded Type Library from the workspace.

The argument to DeleteTypeLib is as follows:

         [1] TypeLib: character vector

The Type Library may be identified by its name or by its class id.

The result is 0, 1 or ¯1.

If successful, the specified Type Library, and all dependant Type Libraries not referenced by any other currently loaded Type Libraries, are removed from the active workspace. The result is 1.

If the specified Type Library is in use, no action is taken and the result is 0.

If the argument is not the name or CLSID of a loaded Type Library, no action is taken and the result is ¯1.

DelRow   Method 154

Applies to   Grid

This method is used to delete a specified row from a Grid object.

The argument to DelRow is a 1 or 2-element array as follows:

[1] Row number: number of the row (integer) to delete

[2] Undo flag: 0 or 1 (optional; default 0)

If the *Undo flag* is 1, the column may subsequently be restored by invoking the Undo method. If the *Undo flag* is omitted or is 0, the operation may not be undone.

Depth   Property

Applies to   TreeView

The Depth property specifies the structure of the items in a TreeView object. It is either a scalar 0 or an integer vector of the same length as the Items property.

A value of 0 indicates that the corresponding item is a top-level item. A value of 1 indicates that the corresponding item is a child of the most recent item whose Depth is 0; a value of 2 indicates that the corresponding item is a child of the most recent item whose Depth is 1, and so forth.

For example:

AIRPORTS	DEPTH	Description
Europe	0	Top-level (root) item
UK	1	1st sub-item of Europe
London Heathrow	2	1st sub-item of UK
London Gatwick	2	2nd sub-item of UK
Manchester	2	3rd sub-item of UK
France	1	2nd sub-item of Europe
Paris CDG	2	1st sub-item of France
Americas	0	Top-level (root) item
USA	1	1st sub-item of N.America
California	2	1st sub-item of USA
Los Angeles	3	1st sub-item of California
San Francisco	3	2nd sub-item of California
East Coast	2	2nd sub-item of USA
New York	3	1st sub-item of East Coast
Kennedy	4	1st sub-item of NY
La Guardia	4	2nd sub-item of NY

```
'F'⎕WC'FORM' 'International Airports'
'F.TV'⎕WC'TreeView'AIRPORTS(0 0)(100 100)('Depth'DEPTH)
```
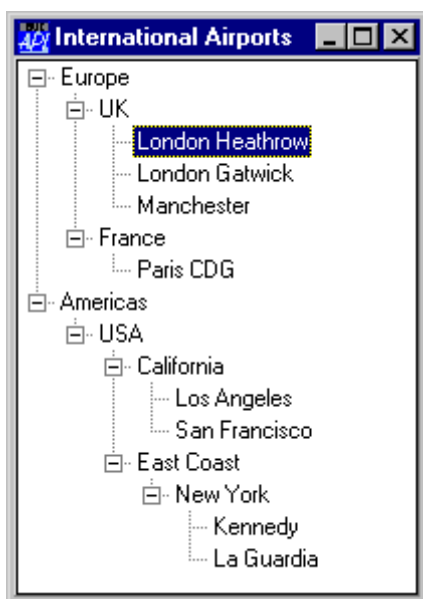
Detach   Method 270

Applies to   ActiveXContainer, ActiveXControl, Animation, Bitmap, BrowseBox,
                     Button, Calendar, Circle, Clipboard, ColorButton, Combo, ComboEx,
                     CoolBand, CoolBar, Cursor, DateTimePicker, Edit, Ellipse, FileBox,
                     Font, Form, Grid, Group, Icon, Image, ImageList, Label, List,
                     ListView, Locator, Marker, MDIClient, Menu, MenuBar, MenuItem,
                     Metafile, MsgBox, OCXClass, OLEClient, OLEServer, Poly, Printer,
                     ProgressBar, PropertyPage, PropertySheet, Rect, RichEdit, Scroll,
                     Separator, SM, Spinner, Splitter, Static, StatusBar, StatusField,
                     SubForm, SysTrayItem, TabBar, TabBtn, TabButton, TabControl,
                     TCPSocket, Text, Timer, TipField, ToolBar, ToolButton,
                     ToolControl, TrackBar, TreeView, UpDown

This method is used to detach the GUI component from an object without losing the
functions, variables and sub-namespaces that it may contain..

The Detach method is niladic.

The effect of this method is to remove the GUI component associated with the named
object, leaving behind a plain namespace of the same name. All non-GUI child objects
are retained. GUI child objects are either destroyed, or similarly converted to plain
namespaces depending upon the values of their KeepOnClose properties.

DevCaps   Property

Applies to   Printer, Root

This property reports the device capabilities of the screen or printer. It is a 3-element nested vector as follows. New elements may be added to DevCaps in a future release.

[1] Height and Width:	2-element numeric vector of device in pixels
[2] Height and Width:	2-element numeric vector of device in mm
[3] Number of colours:	integer scalar

This property is useful if you want to make objects of a specific physical size on a printer. For example, to draw a 10mm square on a Printer *'P'* at (5,5):

```
Size ← 10 × ⊃÷/2↑'.' ⎕WG 'DevCaps'
'P.R' ⎕WC 'Rect' (5 5) Size ('Coord' 'Pixel')
```

Please note that the physical size reported for the screen is typically only a *nominal* size, because, if you use a generic video driver, Windows has no way to tell that you have a 14", 15" or 17" screen attached to your computer.

Directory   Property

Applies to   FileBox

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

If, whilst interacting with the FileBox, the user changes directory and exits by pressing "OK" or by closing the FileBox, the value of the Directory property is updated accordingly.

DisplayChange   Event 137

Applies to   Root

If enabled, this event is reported when the user changes the screen resolution or number of colours. The event is reported for information only; you cannot prevent the change from occurring.

The event message reported as the result of ⎕DQ, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	`'DisplayChange'` or 137
[3] Height:	Integer. Number of pixels in the y-direction
[4] Width:	Integer. Number of pixels in the x-direction
[5] Number of colours:	Integer.

Divider   Property

Applies to   ToolControl

The Divider property controls the presence or absence of a recessed line drawn above, below, to the left of, or to the right of a ToolControl object.

Divider is a single number with the value 0 (dividing line is *not* drawn) or 1 (a dividing line *is* drawn); the default is 1.

Dockable   Property

Applies to   CoolBand, Form, SubForm, ToolControl

The Dockable property specifies whether or not an object may be docked or undocked.

Dockable is a character vector containing *'Never'* (the default), *'Always'* or *'Disabled'*.

If Dockable is *'Never'*, the object may not be docked or undocked by the user, and the docking menu items are not present in the object's context menu. This is the default.

If Dockable is *'Always'*, the object may be docked or undocked by the user, and the docking menu items are present in the object's context menu.

If Dockable is *'Disabled'*, the object may not currently be docked or undocked by the user, but the docking menu items are present in the object's context menu.

Note that by default, the user may switch between Dockable *'Always'* and *'Disabled'* by toggling the *Dockable* menu item. If you want to exercise full control over this property, you may implement your own context menu (see ContextMenu Event)

DockAccept   Event 483

Applies to   CoolBand, CoolBar, Form, SubForm, ToolControl

If enabled, this event is reported by a host object just before it accepts a client object docking operation. This event is reported (by the host) immediately after the DockRequest is reported (by the client).

The event message reported as the result of □*DQ*, or supplied as the right argument to your callback function, is a 7-element vector as follows :

       [1] Host Object: ref or character vector

       [2] Event name or code: *'DockAccept'* or 483

       [3] Client Object: ref or character vector

       [4] Edge: character vector

       [5] y-position: number

       [6] x-position: number

       [7] Outline rectangle: 4-element nested

Elements 4-7 of this event message are the same as those reported by DockMove, and the effect of a callback function is identical. See DockMove for further information.

DockCancel   Event 485

Applies to   CoolBand, CoolBar, Form, SubForm, ToolControl

If enabled, this event is reported by a client object when the user aborts a docking operation by pressing Escape.

The event message reported as the result of □*DQ*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

       [1] Client Object: ref or character vector

       [2] Event name or code: *'DockCancel'* or 485

This event is reported for information only and cannot be cancelled or inhibited in any way.

DockChildren   Property

Applies to   CoolBar, Form, SubForm

The DockChildren property specifies the names of client objects that may be docked in a host object.

DockChildren may be a simple character scalar or vector, or a vector of character vectors. Each item represents the name of an object that may be docked. Notice that you must specify the simple name of the object, excluding any part of its full pathname that refers to a parent; i.e. the specified names must not contain any leading pathname information.

If the name of a dockable object occurs in the DockChildren property, the host object will generate DockMove events when the client is dragged over it, and will generate a DockAccept event when a docking operation takes place.

If the name of the client object is not present in its DockChildren property, the object will not respond in any way as the client is dragged over it.

The following example shows the creation of 3 forms, all of which are dockable in a host form called H1.

The first, *C1*, is a totally independent Form. When docked in *H1*, it will become a SubForm *H1.C1*. When undocked, it will revert to an independent Form *C1*.

The second, *C2*, is created initially as a child of *H1* and will therefore be displayed above it in the window stacking order. When docked it will become a SubForm *H1.C2*. When undocked, it will revert back to a dependant Form *H1.C2*. In all cases, it appears on top of *H1*.

The third, *C3*, is created initially as a child of another Form, *H2*. When docked (in *H1*) it will become a SubForm *H1.C3*. When undocked, it will become a dependant Form *H1.C3*, and will therefore appear above *H1* in the stacking order.

```
'H1'⎕WC'Form' 'Host'

'C1' ⎕WC 'Form' 'Client 1' ('Dockable' 'Always')
'H1.C2' ⎕WC 'Form' 'Client 2' ('Dockable' 'Always')
'H2.C3' ⎕WC 'Form' 'Client 3' ('Dockable' 'Always')

Host.DockChildren←'C1' 'C2 'C3'
```

Docked   Property

Applies to   Form, SubForm

The Docked property is a read-only property that indicates whether or not an object is currently docked.

Docked is a single number with the value 0 (is not docked) or 1 (is docked).

DockEnd   Event 484

Applies to   CoolBand, CoolBar, Form, SubForm, ToolControl

If enabled, this event is reported by a client object after it has been successfully docked in a host object.

The event message reported as the result of ⎕DQ, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1] Client Object:	ref or character vector
[2] Event name or code:	*'DockEnd'* or 484
[3] Original parent:	ref or character vector
[4] New parent	ref or character vector
[5] Flag	Boolean

This event is reported for information only and cannot be cancelled or inhibited in any way.

Flag is 1 if the object was docked; 0 if it was undocked..

DockMove   Event 481

Applies to   CoolBand, CoolBar, Form, SubForm, ToolControl

If enabled, this event is reported by a host object when a dockable object (the client) is dragged over it. The event will only be reported if the name of the client object is included in the list of objects that the host object will accept, which is defined by its DockChildren property.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 7-element vector as follows :

[1] Host Object:	ref or character vector
[2] Event name or code:	*'DockMove'* or 481
[3] Client Object:	ref or character vector
[4] Edge:	character vector
[5] y-position:	number
[6] x-position:	number
[7] Outline rectangle:	(see below)

The 4th element of the event message Edge is a character vector that indicates along which edge of the host object the client object will be docked if the mouse button is released. It is either *'Top'*, *'Bottom'*, *'Left'*, *'Right'* or *'None'*. The latter indicates that the object will not be docked. An object will dock only if the mouse pointer is inside, and sufficiently near to an edge of, the host.

The 5th and 6th elements of the event message report the position of the mouse pointer in the host object.

The 7th element of the event message is a 4-element nested vector containing the y-position, x-position, height and width of a rectangle. If Edge is *'None'*, this is the bounding rectangle of the client object. Otherwise, the rectangle describes a docking zone in the host that the client object will occupy when the mouse button is released.

If a callback function returns 0, the system displays the bounding rectangle and not a docking zone, and the docking operation is inhibited. You could use this mechanism to prohibit docking along one or more edges, whilst allowing it along others.

A callback function may modify the event message to cause a different sized docking zone to be displayed, or to force docking along a particular edge.

The DockMove event is generated repeatedly as the docking object is dragged.

DockRequest   Event 482

Applies to   CoolBand, CoolBar, Form, SubForm, ToolControl

If enabled, this event is reported by a client object just before it is docked in a host object, when the user releases the mouse button.

The event message reported as the result of □*DQ*, or supplied as the right argument to your callback function, is a 7-element vector as follows :

[1] Client Object:	ref or character vector
[2] Event name or code:	`'DockRequest'` or 482
[3] Host Object:	ref or character vector
[4] Edge:	character vector
[5] y-position:	number
[6] x-position:	number
[7] Outline rectangle:	4-element nested

Elements 4-7 of this event message are the same as those reported by DockMove, and the effect of a callback function is identical. See DockMove for further information.

DockShowCaption   Property

Applies to   Form, SubForm

The DockShowCaption property specifies whether or not a Form displays a title bar when it is docked as a SubForm.

DockShowCaption is a single number with the value 0 or 1 (the default).

The DockShowCaption property may be toggled on and off by the user from the object's context menu.

The first picture below illustrates a Form, docked as a SubForm, whose DockShowCaption property is 1, but is about to be set to 0.
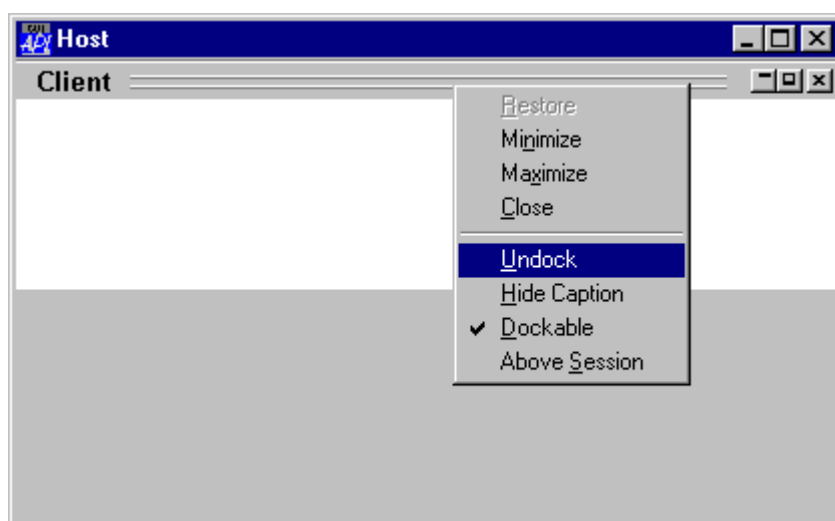
The next picture shows the same docked Form with DockShowCaption set to 0.
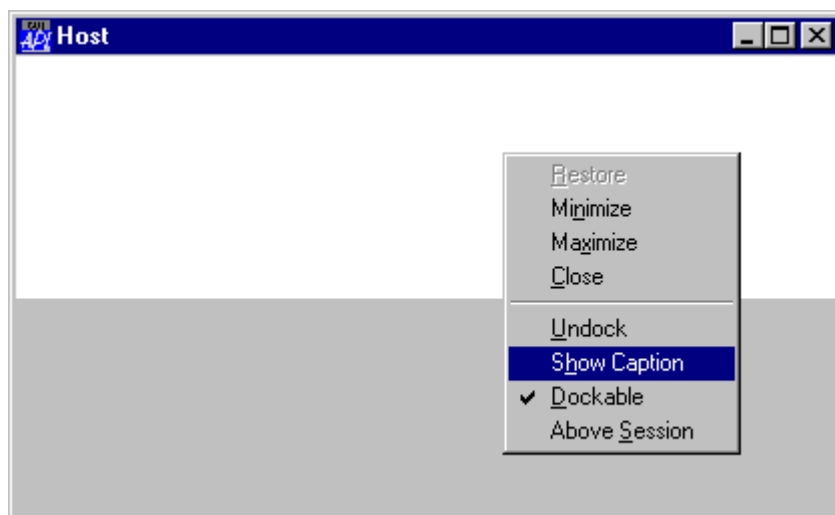
DockStart   Event 480

Applies to   CoolBand, CoolBar, Form, SubForm, ToolControl

If enabled, this event is reported by a dockable object (one whose Dockable property is set to 1) when the user starts to drag it using the mouse.

The event message reported as the result of ⎕DQ, or supplied as the right argument to your callback function, is a 2-element vector as follows :

      [1] Object: ref or character vector

      [2] Event name or code: `'DockStart'` or 480

A callback function may prevent the docking operation from starting by returning 0.

Dragable   Property

Applies to   ActiveXControl, Animation, Button, Calendar, Circle, ColorButton, Combo, ComboEx, DateTimePicker, Edit, Ellipse, Grid, Group, Image, Label, List, ListView, Marker, Poly, ProgressBar, Rect, RichEdit, Scroll, SM, Spinner, Static, StatusField, Text, TrackBar, TreeView, UpDown

This property determines whether or not an object may be the subject of a "drag and drop" operation. It is a single number with the value 0, 1 or 2. A value of 0 (which is the default) means that the object may not be drag/dropped. A value of 1 means that the object may be drag/dropped and that during the "drag" operation, a box representing the bounding rectangle around the object is displayed on the screen. A value of 2 means that the **outline** of the object is displayed as the object is dragged, or, if the object is an Image with a Bitmap property containing the name of an Icon object, the icon itself is displayed as it is dragged. For rectangular non-graphical objects, values of 1 and 2 are equivalent.

If an object is Dragable, the user may drag it by positioning the mouse pointer within the object, depressing the left mouse button, then moving the mouse with the button held down. During the drag operation, the mouse pointer is automatically changed to a "drag" symbol. The object is "dropped" by releasing the left mouse button. The effect depends upon where it is dropped, and on the action associated with the DragDrop event for the object under the mouse pointer (if any).

If there is no object under the mouse pointer, the "drag and drop" operation is ignored. Otherwise, the object under the mouse pointer generates a DragDrop event.

If the object under the mouse pointer is the parent of the object that has been "dragged and dropped", the default action is for the system to move that object to the new location within its parent. If you wish to allow your user to freely move an object within its parent Form or Group, simply set its Dragable property to 1; the system will take care of the rest. If you want to allow the user to move an object, but you want to know about it when it happens, you can associate a callback function to the DragDrop event that queries the new position. To permit the operation to complete, the callback function should either not return a result or it should return something other than a scalar 0. To selectively disable movement, your callback function should return a scalar 0 in circumstances when the "drop" is not to be permitted.

If the object under the mouse pointer is not the parent of the object being dragged, the default action is for the system to ignore the operation. However, by enabling the DragDrop event, your application can of course take whatever action is appropriate, including perhaps moving the dragged object to a new parent.

Note that a Dragable object does not generate a Configure (31) event when it is dragged and dropped.

DragDrop   Event 11

Applies to   ActiveXControl, Animation, Button, Calendar, Circle, ColorButton, Combo, ComboEx, CoolBar, DateTimePicker, Edit, Ellipse, Form, Grid, Group, Image, Label, List, ListView, Marker, MDIClient, Poly, ProgressBar, PropertyPage, Rect, RichEdit, Scroll, SM, Spinner, Static, StatusBar, SubForm, TabBar, Text, ToolBar, ToolControl, TrackBar, TreeView, UpDown

If enabled, this event is reported when the user drops one object over another. It is generated by the object which is being dropped **on**.

The event message reported as the result of $\Box DQ$, or supplied as the right argument to your callback function, is a 7-element vector as follows :

[1] Object:	ref or character vector (target object)
[2] Event code:	*'DragDrop'* or 11
[3] Object:	ref or character vector (dragged object)
[4] Y:	y-position of mouse pointer
[5] X:	x-position of mouse pointer
[6] H:	height of dragged object
[7] W:	width of dragged object
[8] Shift State:	numeric scalar containing the sum of the values associated with the Shift(1), Ctrl(2) and Alt(4) keys when the object was dropped.

Y, H, X and W are reported relative to the object being dropped **on**.

DragItems   Property

Applies to   ListView

The DragItems property is Boolean and specifies whether or not the items in a ListView object may be drag/dropped by the user. Its default value is 1.

DrawMode   Property

Applies to   Circle, Ellipse, Marker, Poly, Rect, Text

The DrawMode property provides direct control over the low-level drawing operation performed by graphical objects.

The DrawMode property specifies the current foreground mix mode. The Windows GDI uses the foreground mix mode to combine pens and interiors of filled objects with the colours already on the screen. The foreground mix mode defines how colours from the brush or pen and the colours in the existing image are to be combined.

DrawMode affects every drawing operation performed by Dyalog APL and not just the initial drawing operation when the object is created. Many of the drawing modes are additive (the result depends not just on what is being drawn, but on what is already there) and the effects may therefore vary. For this reason, DrawMode should normally be used only with unnamed graphical objects.

You could use DrawMode to move or animate graphical objects in circumstances where the standard Dyalog APL behaviour was not ideal.

DrawMode is an integer with one of the following values:

Value	Name	Resulting Pixel Colour
1	R2_BLACK	Pixel is always 0.
2	R2_NOTMERGEPEN	Pixel is the inverse of the R2_MERGEPEN color.
3	R2_MASKNOTPEN	Pixel is a combination of the colors common to both the screen and the inverse of the pen.
4	R2_NOTCOPYPEN	Pixel is the inverse of the pen color.
5	R2_MASKPENNOT	Pixel is a combination of the colors common to both the pen and the inverse of the screen.
6	R2_NOT	Pixel is the inverse of the screen color.
7	R2_XORPEN	Pixel is a combination of the colors in the pen and in the screen, but not in both.
8	R2_NOTMASKPEN	Pixel is the inverse of the R2_MASKPEN color.
9	R2_MASKPEN	Pixel is a combination of the colors common to both the pen and the screen.
10	R2_NOTXORPEN	Pixel is the inverse of the R2_XORPEN color.
11	R2_NOP	Pixel remains unchanged.
12	R2_MERGENOTPEN	Pixel is a combination of the screen color and the inverse of the pen color.
13	R2_COPYPEN	Pixel is the pen color.
14	R2_MERGEPENNOT	Pixel is a combination of the pen color and the inverse of the screen color.
15	R2_MERGEPEN	Pixel is a combination of the pen color and the screen color.
16	R2_WHITE	Pixel is always 1.

DropDown   Event 45

Applies to   ColorButton, Combo, ComboEx, DateTimePicker, Menu

If enabled, this event is reported when the user clicks the drop-down button in a ColorButton, Combo, ComboEx, DateTimePicker or Menu object, just before the drop-down colour selection box, list, calendar or -menu is displayed.

For a DateTimePicker this event only applies if the Style of the DateTimePicker is *'Combo'*.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1] Object: ref or character vector

[2] Event name or code: *'DropDown'* or 45

This event is reported for information only and cannot be disabled or modified in any way.

DropFiles   Event 450

Applies to   ActiveXControl, Animation, Button, Calendar, ColorButton, Combo, ComboEx, CoolBar, DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, ProgressBar, PropertyPage, RichEdit, Scroll, Spinner, Static, StatusBar, SubForm, TabBar, ToolBar, ToolControl, TrackBar, TreeView, UpDown

If enabled, this event is reported when the user drags a file icon or a set of file icons and drops them onto the object. The system takes no action other than to report the event.

The event message reported as the result of *⎕DQ*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	*'DropFiles'* or 450
[3] Files	Vector of character vectors containing the file names.
[4] Item number:	Integer. The index of the item within the object onto which the file(s) was dropped. Applies only to objects that have an Items property such as List, ListView and TreeView.
[5] Shift state:	Integer. Sum of 1=shift key, 2=Ctrl key, 4=Alt key

DropObjects Event 455

Applies to ActiveXControl, Animation, Button, Calendar, ColorButton, Combo, ComboEx, CoolBar, DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, ProgressBar, PropertyPage, RichEdit, Scroll, Spinner, Static, StatusBar, StatusField, SubForm, TabBar, TabBtn, ToolBar, ToolControl, TrackBar, TreeView, UpDown

If enabled, this event is reported when the user drags an object icon or a set of object icons from the Explorer tool (which is part of the Dyalog APL Session) and drops them onto the object. The system takes no action other than to report the event. You can use this event to extend drag-drop functionality in your Session. For example, you could perform an operation by drag-dropping an APL object icon onto a Button in the Session toolbar.

The event message reported as the result of □DQ, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1] Object:		ref or character vector
[2] Event name or code:		*'DropObjects'* or 455
[3] Objects		Vector of character vectors containing the object names.
[4] Item number:		Integer. The index of the item within the object onto which the file(s) was dropped. Applies only to ListView and TreeView. Otherwise this value is ¯1.
[5] Shift state:		Integer. Sum of 1=Shift key, 2=Ctrl key, 4=Alt key

Duplex   Property

Applies to   Printer

Specifies whether pages are printed on separate sheets or back-to-back.

Duplex is a character vector which is either empty or contains *'Simplex'*, *'Vertical'*, or *'Horizontal'*.

The default value for Duplex is derived from the current printer setting and *'Vertical'* and *'Horizontal'* are only effective if the printer supports a duplex capability.

DuplicateColumn   Method 178

Applies to   Grid

This method is used to duplicate a column in a Grid object.

The argument to DuplicateColumn is a 2, 3, 4 or 5-element vector as follows:

[1] Source Column number:	number of the column (integer) to be duplicated
[2] Target Column number:	new column number (integer)
[3] Comment flag:	0 or 1 (optional, default 1)
[4] Lock flag:	0 or 1 (optional, default 1)
[5] Undo flag:	0 or 1 (optional; default 0)

If the *Comment flag* is 1 (the default), any Comments associated with cells in the source column are duplicated in the target column.

If the *Lock flag* is 1 (the default), the lock state of the column is duplicated; otherwise, the new column is not locked.

If the *Undo flag* is 1, the column may subsequently be restored by invoking the Undo method. If this element is omitted or is 0, the operation may not be undone.

DuplicateRow   Method 177

Applies to   Grid

This method is used to duplicate a column in a Grid object.

The argument to DuplicateRow is a 2, 3, 4 or 5-element vector as follows:

[1] Source Row number:	number of the row (integer) to be duplicated
[2] Target Row number:	new row number (integer)
[3] Comment flag:	0 or 1 (optional, default 1)
[4] Lock flag:	0 or 1 (optional, default 1)
[5] Undo flag:	0 or 1 (optional; default 0)

If the *Comment flag* is 1 (the default), any Comments associated with cells in the source row are duplicated in the target column.

If the *Lock flag* is 1 (the default), the lock state of the row is duplicated; otherwise, the new row is not locked.

If the *Undo flag* is 1, the row may subsequently be restored by invoking the Undo method. If this element is omitted or is 0, the operation may not be undone.

DyalogCustomMessage1   Event 95

Applies to   Form

This event allows external applications and dynamic link libraries to insert events into the Dyalog APL/W message queue.

DyalogCustomMessage1 may be invoked from a C program as follows:

```
msg=RegisterWindowMessage("DyalogCustomMessage1");
SendMessage(hWnd,msg,wParam,lParam);
```

where hWnd is the window handle of the object in the Dyalog APL Workspace. If the object is a Form, this may be obtained using FindWindow(). If not, hWnd may be passed to the external process as an argument to a function.

The parameters wParam and lParam are reported as numeric arguments to the APL callback function.

NOTE: It is not possible to pass pointers to data in wParam or lParam. When the APL callback executes the pointers may not be valid.

If a callback function is attached to the event, the callback function will be run when the event reaches the top of the queue.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to ¯1 or by returning 0 from a callback function.

The result of a callback function is not returned to the external application.

The event message reported as the result of *□DQ*, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	'*DyalogCustomMessage1*' or 95
[3] wparam:	integer
[4] lparam:	integer

EdgeStyle   Property

Applies to   ActiveXControl, Animation, Button, Calendar, ColorButton, Combo,
ComboEx, DateTimePicker, Edit, FileBox, Form, Grid, Group,
Image, Label, List, ListView, MDIClient, Menu, MenuBar,
MenuItem, MsgBox, Printer, ProgressBar, PropertyPage,
PropertySheet, Rect, RichEdit, Root, Scroll, Separator, SM, Spinner,
Static, StatusBar, StatusField, SubForm, TabBtn, ToolBar, TrackBar,
TreeView, UpDown

This property is used to give a 3-dimensional appearance to screen objects. This is
achieved by drawing the object with a grey or white background colour and by drawing
a border around it using various combinations of black, white and dark grey lines. Note
that this border is drawn *outside* a control but *inside* a Form or SubForm. The value of
the EdgeStyle property is a character vector chosen from the following:

EdgeStyle Value	Description
'None'	Object is drawn with no 3-dimensional effects and the EdgeStyle properties of its children are ignored (treated as *None*)
'Plinth'	Object is drawn with a light shadow along its top and left edges and a dark shadow along its bottom and right edges. This gives the illusion of a raised effect
'Recess'	Object is drawn with a dark shadow along its top and left edges and a light shadow along its bottom and right edges. This gives the illusion of a sunken effect.
'Groove'	Object is drawn with a border that has the appearance of a groove.
'Ridge'	Object is drawn with a border that has the appearance of a ridge.
'Shadow'	Object is drawn with a dark border line along its top and left edges.
'Default'	Object itself is drawn with no 3-dimensional border, but the values of the EdgeStyle properties of its children are observed.
'Dialog'	Used in conjunction with (*'Border'* 2), this gives a Form the appearance of a standard 3-dimensional dialog box. This setting applies **only** to a Form or to a SubForm

The following illustration shows the result obtained using different values of the EdgeStyle property with a Rect object.
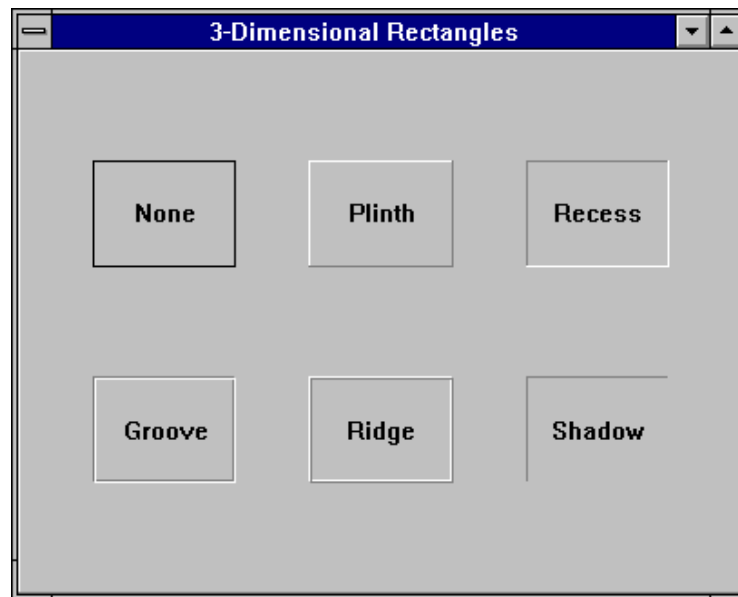
For the Root object, the EdgeStyle property may be '*None*' or '*Default*'. If EdgeStyle is '*None*', screen objects are drawn without 3-dimensional effects of any kind and the value of their EdgeStyle property is ignored. If EdgeStyle is '*Default*', the circle in a Radio Button and the check box in a Check Button are recessed. In addition, all controls are drawn using their default EdgeStyle properties.

Thus setting EdgeStyle on Root to '*None*' causes applications written using Dyalog APL/W Version 6 to appear unchanged. If you have implemented your own 3-dimensional effects, this may be desirable, at least as a temporary measure.

Note that MsgBox, FileBox and the set-up dialog box associated with the Printer object are all drawn with 3-dimensional effects regardless of the value of EdgeStyle on Root. These objects do not have their own EdgeStyle properties.

If you set EdgeStyle to '*None*' on the Root object, all your objects will (by default) be drawn without 3-dimensional effects.

The EdgeStyle property is generally less applicable and behaves differently in Version 8. In particular, EdgeStyle '*None*' is not honoured for objects which have a natural built-in 3-dimensional appearance in Windows 95 and NT.

Edit   Object

Purpose	Allows user to enter or edit data.
Parents	ActiveXControl, CoolBand, Form, Grid, Group, PropertyPage, SubForm, ToolBar, ToolControl
Children	Circle, Cursor, Ellipse, Font, Marker, Poly, Rect, Text, Timer
Properties	Type, Text, Posn, Size, Style, Coord, Border, Justify, Active, Visible, Event, VScroll, HScroll, SelText, Sizeable, Dragable, FontObj, FCol, BCol, CursorObj, AutoConf, Data, Attach, TextSize, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, FieldType, MaxLength, Decimals, Password, ValidIfEmpty, ReadOnly, FormatString, Changed, Value, Translate, Accelerator, AcceptFiles, WantsReturn, KeepOnClose, Redraw, MethodList, ChildList, EventList, PropList
Events	BadValue, Change, Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, FontCancel, FontOK, GotFocus, Help, KeyError, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Select
Methods	Animate, ChooseFont, Detach, GetFocus, GetTextSize, ShowSIP

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

Single-Line Edit

The FieldType property (which applies only to a single-line Edit object) is either an empty vector (the default) or specifies the type of the field. If FieldType is empty, the object is a simple character field whose contents are defined by its Text property which is a character vector. If FieldType is `'Numeric'`, `'LongNumeric'`, `'Currency'`, `'Date'`, `'LongDate'`, or `'Time'` the object contains a number defined by the Value property. For fields of these types, basic validation is provided during user input. The field is revalidated when the user attempts to "leave" it and at this point the object will generate a BadValue event if its contents are inconsistent with its FieldType.

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

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

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

The picture below illustrates a standard single-line edit field and one with EdgeStyle set to `'Plinth'`. The second field is also smaller than the text it occupies and may be scrolled.
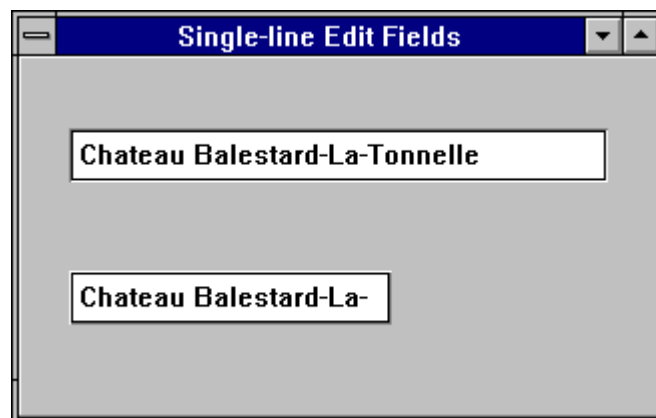
Multi-Line Edit

If the Style is `'Multi'`, Text may be a simple character vector, a matrix, or a vector of vectors. If you specify Text as a matrix, "new-line" characters are automatically added at the end of each row. Similarly, if you specify Text as a vector of vectors, "new-line" characters are added after each vector. The user may insert a "new-line" character in the text by pressing Ctrl-Enter. When you query the Text property using `⎕WG`, its contents are returned in the same form that it was specified, or if it was not specified, as a matrix. "New-line" characters are not returned, but are used to delimit rows (of a matrix) or vectors (of a vector of vectors). However, if you specified Text as a simple vector (or an empty vector) and the user has added a "new-line", Text will be returned by `⎕WG` as a vector of vectors.

The behaviour of the Enter key is defined by the WantsReturn property. If this is 0 (the default) the Enter key is ignored by the Edit object and may instead generate a Select event on a Button. In this case the user must press Ctrl+Enter to input a new line. If WantsReturn is 1, the Enter key inputs a new line into the Edit object.

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

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

The VScroll property determines whether or not data may be scrolled vertically and whether or not the object has a vertical scrollbar. A value of `0` inhibits scrolling; `¯2` means scrollable, without a scrollbar; `¯1` means scrollable with a scrollbar.

The picture below illustrates two different multi- line edit boxes. The box on the left is defined with (`'Style' 'Multi'`) (`'HScroll' ¯2`). Notice that the fourth wine, "Chateau Balestard-La-Tonnelle", is only partially visible. It may be viewed by scrolling sideways with the cursor keys. If HScroll had been set to ¯1 a horizontal scrollbar would also be present. The box on the right is defined as with(`'Style' 'Multi'`) (`'VScroll' ¯1`)(`'Justify' 'Centre'`)

The setting of Justify forces word-wrapping. Notice how the fourth wine, "Chateau Balestard-La-Tonnelle", is spread over two lines.
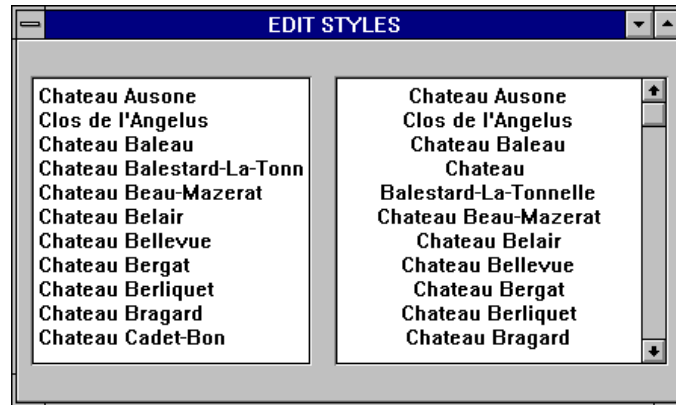
The SelText property identifies the portion of the text that is selected. It may be used to pre-select (and highlight) a part of the text, or to report the part of the text selected by the user. SelText is a 2-element integer vector which specifies the start and end of the selected area. Its structure depends upon the nature of the data specified by Text. See the description of SelText for details.

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

EditImage   Property

Applies to   ComboEx

Specifies whether or not the edit control portion of the ComboEx displays an image for selected items.

EditImage is a single number with the value 0 or 1 (the default). If EditImage is 1, the image associated with the selected item is displayed in the edit control, portion of the ComboEx object, to the left of the text. If EditImage is 0, only the item text is displayed in the edit control.

EditImageIndent   Property

Applies to   ComboEx

Specifies whether or not the indents associated with items in a ComboEx object are honoured in the edit control portion of the ComboEx.

EditImageIndent is a single number with the value 0 or 1 (the default).

If EditImageIndent is 1, the selected item which is displayed in the edit control portion of the ComboEx object is indented in the same way as when it is displayed in the dropdown portion of the object. The amount of indentation is specified by the Indents property.

If EditImageIndent is 0, the item displayed in the edit control portion of the ComboEx is not indented.

EditLabels   Property

Applies to   ListView, TreeView

The EditLabels property is Boolean and specifies whether or not the labels (specified by the Items property) in a ListView or TreeView object may be edited by the user. Its default value is 0 (editing is not allowed).

If EditLabels is 1, the user begins editing by clicking the label of the item that has the focus. This causes a pop-up edit box to appear around the item and allows the use to change it. A BeginEditLabel event is reported at the start of the edit operation and an EndEditLabel event is reported on its completion. You may control the edit of a particular label using callback functions attached to these events.

Ellipse   Object

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

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

Children   Timer

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

Events   Close, Create, DragDrop, Help, MouseDblClick, MouseDown, MouseMove, MouseUp, Select

Methods   Detach

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

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

The Size property specifies the height and width of each bounding rectangle, measuring away from the origin. To obtain a perfect circle, you must take the **aspect ratio** of the device into account. This is available from the DevCaps property of the Root and Printer objects. Alternatively you can use the Circle object.

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

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

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

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

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

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

Single Ellipse, Circle or Arc

For a single ellipse, circle or arc, Points is a 2-element vector which specifies the y-coordinate and x-coordinate of the top-left corner of the bounding rectangle. Size is also a simple 2-element vector whose elements specify the height and width of the bounding rectangle. LStyle and LWidth are both simple scalar numbers.

FStyle is either a single number specifying a standard fill pattern, or the name of a Bitmap object which is to be used to fill the ellipse, circle or arc. FCol, BCol and FillCol are each either single numbers representing standard colours, or 3-element vectors which specify colours explicitly in terms of their RGB values.

Examples

First make a Form :

```
'F' ⎕WC 'Form'
```

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

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

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

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

Ditto, but between 45 and 135 degrees :

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

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

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

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

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

Multiple Ellipses, Circles or Arcs

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

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

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

If Start is specified, but not End, end angles default to ($^{-}1\downarrow+\backslash Start$),$\supset 2$. If End is specified, but not Start, start angles default to $0,^{-}1\downarrow+\backslash End$

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

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

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

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

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

Examples

First make a Form :

```
'F' ⎕WC 'Form'
```

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

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

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

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

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

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

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

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

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

End   Property

Applies to   Circle, Ellipse

This property specifies one or more end-angles for an arc, pie-slice, or chord of a circle or ellipse. It may be used in conjunction with Start which specifies start angles. Angles are measured counter-clockwise from the x-axis at the centre of the object.

If a single arc is being drawn, End is a single number that specifies the end angle of the arc in radians (`0 -> ○2`). If multiple arcs are being drawn, End is either a single number as before (the end angle for several concentric arcs) or a numeric vector with one element per arc.

If Start is not specified, the default value of End is `⊃2`. Otherwise, the default value of End is (`(¯1↓+\Start), ○2`).

EndEditLabel   Event 301

Applies to   ListView, TreeView

If enabled, this event is reported when the user signals completion of an edit operation in a ListView or TreeView object. This occurs when the item being edited loses the focus or when the user presses the Enter key. The default processing for the event is to update the item label (string) with the edited text in the pop-up edit box.

You may disable the update operation by setting the action code for the event to `¯1`. You may also prevent the update from occurring by returning 0 from a callback function. You may specify the text used to update the item by returning the event message (containing the desired text) from a callback function. Finally, you may change the text of any item dynamically by calling EndEditLabel as a method.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	`'EndEditLabel'` or 301
[3] Item number:	Integer. The index of the item.
[4] Text:	character vector containing the text that will be used to update the item's label.

EndSplit   Event 282

Applies to   Splitter

If enabled, this event is reported when the user releases the left mouse button to signify the end of a drag operation on a Splitter object.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to ¯1 or by returning 0 from a callback function.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	'*EndSplit*' or 282
[3] Y:	y-position of top left corner
[4] X:	x-position of top left corner
[5] H:	height of the Splitter
[6] W:	width of the Splitter

See also StartSplit, Splitting.

EnterReadOnlyCells Property

Applies to Grid

This is a Boolean property that specifies whether or not the user may visit read-only cells in a Grid object. Its default value is 1.

In this context, a *read-only* cell is one that satisfies one or more of the following conditions:

- it has no associated Input object
- its associated Input object is a Label
- its associated Input object is an Edit object with ReadOnly set to 1.
- its associated Input object is inactive (Active 0)

If EnterReadOnlyCells Cells is set to 0 and the user clicks the mouse on a *read-only* cell, the current cell does not change although CellDown, CellUp and CellDblClick events are reported if enabled. If the user presses a cursor movement key that would otherwise cause the cursor to move into a *read-only* cell, the cursor moves instead to the nearest *editable* cell in the appropriate direction.

Event   Property

Applies to   ActiveXContainer, ActiveXControl, Animation, Bitmap, BrowseBox, Button, Calendar, Circle, Clipboard, ColorButton, Combo, ComboEx, CoolBand, CoolBar, Cursor, DateTimePicker, Edit, Ellipse, FileBox, Form, Grid, Group, Icon, Image, ImageList, Label, List, ListView, Locator, Marker, MDIClient, Menu, MenuBar, MenuItem, Metafile, MsgBox, OCXClass, OLEClient, OLEServer, Poly, Printer, ProgressBar, PropertyPage, PropertySheet, Rect, RichEdit, Root, Scroll, Separator, SM, Spinner, Splitter, Static, StatusBar, StatusField, SubForm, SysTrayItem, TabBar, TabBtn, TabButton, TabControl, TCPSocket, Text, Timer, TipField, ToolBar, ToolButton, ToolControl, TrackBar, TreeView, UpDown

This property defines how an object responds to user actions. Unlike other properties which only have a single value, this property has a value corresponding to each of the events that may be generated by a particular object. Each event setting is specified by a 2 or 3-element vector containing :

[1]	event name(s) (optionally prefixed by the string `'on'`) or event number(s) (numeric scalar or vector).
[2]	action code (numeric scalar or character vector)

¯1	inhibit (ignore) event
0	handle event, do not report to APL
1	handle event, then report to APL
fn	name of callback function to be executed
fn&	name of callback function to be executed *asynchronously*
⍎expr	expression to be executed

[3]	any array (optional)

The event number 0 and the event name `'All'`, apply to all events supported by that object.

Like any other property, the Event property can be set using assignment. However, certain special considerations apply which are discussed later.

If *action* is set to ¯1, the event is ignored by APL. If, for example, you set the action on a KeyPress event to ¯1, all keystrokes for the object in question will be ignored. Similarly, if you set the *action* on a Close event for a Form to ¯1, the user will be unable to close the Form. This is possible because APL intercepts most events before Windows itself takes any action. However, certain events (e.g. focus change events) are not notified to APL until **after** the event has occurred and **after** Windows has itself responded in some way. In these circumstances it is not always practical for APL to undo what Windows has already done, and an action code of ¯1 is treated as if it were 0. For further details, see the individual entries for each event type in this Chapter.

If action code is set to 0 (the default), the event is processed by APL and Windows in the normal way (this is referred to herein as "default processing") but your program is not notified in any way that the event has occurred. For example, the default processing for a keystroke is to action it (via the translate table) and either echo a character in the object or perform some other appropriate function.

If action code is set to 1, the event is first processed by APL (and Windows) in the normal way, then `⎕DQ` terminates, returning an **event message** as its result. The format of the event message is given under the description of each event type in this Chapter.

If action code is set to a character vector that specifies the name of a function, this function (termed a "callback") will be executed automatically by `⎕DQ` whenever the event occurs. Default processing of the event is deferred until after the callback has been run, and may be inhibited or modified by its result. If the callback function returns no result, or returns a scalar 1, normal processing of the event is allowed to continue as soon as the callback completes. If the callback returns a scalar 0, normal processing of the event is inhibited and the effect is identical to setting the action code to ¯1. A callback function may also return an event message as its result. If so, `⎕DQ` will action *this* event rather than the original one that fired the callback.

Note that that if a callback function does not exist at the instant it is invoked, `⎕DQ` terminates with a *VALUE ERROR*. However, the name of the missing function is reported in the Status Window.

If, in setting the Event property, the event name is prefixed by the string `'on'`, for example, `'onSelect'`, the right argument to the callback function will contain one or more object *references*. If not, it will contain the corresponding object *names* or, if the object has no name, its display form.

If the character & is appended to the name of a callback function, the callback is executed **asynchronously** in a new thread. In this case, default processing of the event is performed immediately. Such a callback should not return a result; if it does so, it will be treated as normal output and will therefore be displayed in the Session window.

When a callback function is invoked by $\square DQ$, the corresponding event message is supplied as its right argument. The format of the event message is given under the description of each event type in this chapter. Note that the first element of the event message is always a *reference to* or the *name of* the object that generated the event. It is a *reference* if the event name was prefixed by the string `'on'`; otherwise it is a character vector containing the object's name.

If an array was specified as the 4th element of the value of the Event property, the value of this array is supplied as its left argument.

Notice that $\square DQ$ takes account of the syntax defined for the callback, and supplies these arguments only if it is appropriate to do so. It is possible therefore to use a niladic callback function. This is appropriate if the callback can perform its task without needing to interrogate the event message.

If action code is set to a character vector whose first element is the execute symbol (⍎) the remaining string will be executed automatically whenever the event occurs. The default processing for the event is performed first and may not be changed or inhibited in any way.

Notice that when you specify the action to be taken on the occurrence an event there is a great difference between `'FOO'` and `'⍎FOO'`. The former causes APL to invoke the function *FOO* as a *callback function*. If the function takes an argument, APL will supply it with the event message. Secondly, the result (if any) of the function *FOO* will be used by APL and may cause the event to be disabled or changed in some way. In the second case, APL will perform the default processing for the event and then execute *FOO* without supplying an argument. If the function returns a result, it will be displayed in the Session.

When using $\square WC$ and $\square WS$ to assign different events to different callbacks, it is not necessary to repeat the `'Event'` keyword. Instead, several event settings can be specified at once. In any given occurrence of the Event property you may use event number(s) or event name(s); however you may not mix numbers and names together.

If you use event names, your callback functions will receive event names in their right argument when invoked. That is to say that the second element of the event message will be a character vector. If you use event numbers, the second element of the event message will be numeric. If you want to specify several event names at once, you must enclose them. If you use numbers, $\square WC$ and $\square WS$ are more tolerant about the **structure** of their arguments, and will accept many different expressions.

If no events are set, the result obtained by `⎕WG` and the result obtained by referencing Event directly are different:

```
      'F'⎕WC'Form'
      DISPLAY 'F'⎕WG'Event'
.→--.
|0 0|
'~--'
      DISPLAY F.Event
.⊖-----------.
| .→-------. |
| | .⊖. .⊖. | |
| | | | | | | |
| | '-' '-' | |
| '∊-------' |
'∊-----------'
```

Asynchronous Callback Functions

If you append the character `&` to the name of the callback function in the *Event* specification, the callback function will be executed asynchronously in a new thread when the event occurs. If not, it is execute synchronously.

For example, the event specification:

```
      onSelect←'DoIt&'
```

tells `⎕DQ` to execute the callback function *DoIt* asynchronously as a thread when a Select event occurs on the object. Note that a callback function executed in this way should not return a result (because `⎕DQ` does not wait for it) and any result will be displayed in the Session window.

Specifying the Event property using Assignment

There are two ways to specify the Event property using assignment; you can specify the entire set of events, or you can set individual events one by one.

To specify the entire set of events, the array assigned to Event must contain one or more nested vectors, each containing 2 or 3 elements as described above. For example, if *F1* is a Form:

Invoke callback function *FOO* on MouseDown, the first element of the right argument to *FOO* will contain a *namespace reference* to *F1*. All other events perform their default actions.

```
F1.Event ← 'onMouseDown' 'FOO'
```

Invoke callback function *FOO* on MouseDown, the first element of the right argument to *FOO* will contain the *character vector* '*F1*'. All other events perform their default actions.

```
F1.Event ← 'MouseDown' 'FOO'
```

Invoke callback function *FOO* on MouseDown and MouseUp. All other events perform their default actions.

```
F1.Event ← ('onMouseDown' 'FOO')( 'onMouseUp' 'FOO')
```

Add callback function *FOO* with ('*THIS*' 1) as its left-argument on the MouseMove event. All other events perform their default actions.

```
F1.Event, ← ⊂ 'onMouseMove' 'FOO' ('THIS' 1)
```

To set individual events one by one, you make the assignment to the event name prefixed by the string '*on*'. In all cases, the first element of the right argument to *FOO* will contain a *namespace reference* to *F1*. You must use the '*on*' prefix; you cannot assign to the Event name itself.

Invoke callback function *FOO* on MouseDown.

```
F1.onMouseDown ← 'FOO'
```

Add the same callback for MouseUp.

```
F1.onMouseUp ← 'FOO'
```

Add callback function *FOO* with ('*THIS*' 1) as its left-argument on the MouseMove event.

```
F1.onMouseMove ← 'FOO' ('THIS' 1)
```

Specifying the Event property using □*WC* and □*WS*

Examples using Event Names (□*WS*)

Ignore MouseDown (1) event (APL will perform the default processing for you)

```
'F1' □WS 'Event' 'MouseDown' 0
```

Terminate □*DQ* on MouseDown

```
'F1' □WS 'Event' 'MouseDown' 1
```

Invoke callback function *FOO* on MouseDown, the first element of the right argument to *FOO* will contain a *namespace reference* to *F1*.

```
'F1' □WS 'Event' 'onMouseDown' 'FOO'
```

Invoke callback function *FOO* on MouseDown, the first element of the right argument to *FOO* will contain the *character vector* '*F1*'.

```
'F1' □WS 'Event' 'MouseDown' 'FOO'
```

Invoke callback function *FOO* on MouseDown and MouseUp

```
'F1' □WS 'Event' ('onMouseDown' 'onMouseUp') 'FOO'
```

Invoke callback function *FOO* with ('*THIS*' 1) as its left-argument on MouseDown

```
'F1' □WS 'Event' 'onMouseDown' 'FOO' ('THIS' 1)
```

Invoke callback function *FOO* with ('*THIS*' 1) as its left-argument on MouseDown, MouseUp and MouseMove

```
EV ← 'onMouseDown' 'onMouseUp' 'onMouseMove'
'F1' □WS 'Event' EV 'FOO' ('THIS' 1)
```

Execute the expression *COUNT* +←1 on MouseDown

```
'F1' □WS 'Event' 'MouseDown' '⍎COUNT+←1'
```

Execute the expression *COUNT* +←1 on MouseDown, MouseUp and MouseMove

```
EV ← 'MouseDown' 'MouseUp' 'MouseMove'
'F1' □WS 'Event' EV '⍎COUNT+←1'
```

Examples using Event Numbers (⎕WS)

Ignore MouseDown (1) event (APL will perform the default processing for you)

```
'F1' ⎕WS 'Event' (1 0)
'F1' ⎕WS 'Event' 1 0     ⍝ Ditto
```

Terminate ⎕DQ on MouseDown

```
'F1' ⎕WS 'Event' (1 1)
'F1' ⎕WS 'Event' 1 1     ⍝ Ditto
```

Call function FOO on MouseDown

```
'F1' ⎕WS 'Event' (1 'FOO')
'F1' ⎕WS 'Event' 1 'FOO'    ⍝ Ditto
```

Call function FOO on MouseDown and MouseUp

```
'F1' ⎕WS 'Event' ((1 2) 'FOO')
'F1' ⎕WS 'Event' (1 2) 'FOO'            ⍝ Ditto
'F1' ⎕WS 'Event' 1 2 'FOO'              ⍝ Ditto
'F1' ⎕WS 'Event' (1 'FOO')(2 'FOO')     ⍝ Ditto
```

Call function FOO with ('THIS' 1) as its left-argument on MouseDown

```
'F1' ⎕WS 'Event' (1 'FOO' ('THIS' 1))
'F1' ⎕WS 'Event' 1 'FOO' ('THIS' 1)     ⍝ Ditto
```

Call function FOO with ('THIS' 1) as its left-argument on MouseDown and MouseUp

```
'F1' ⎕WS 'Event' ((1 2) 'FOO' ('THIS' 1))
'F1' ⎕WS 'Event' (1 2) 'FOO' ('THIS' 1)   ⍝ Ditto
'F1' ⎕WS 'Event' 1 2 'FOO' ('THIS' 1)     ⍝ Ditto
'F1' ⎕WS 'Event' 1 2 'FOO' ('THIS' 1)     ⍝ Ditto
```

Execute the expression COUNT +←1 on MouseDown

```
'F1' ⎕WS 'Event' 1 '⍎COUNT+←1'
```

Execute the expression COUNT +←1 on MouseDown, MouseUp and MouseMove

```
'F1' ⎕WS 'Event' (1 2 3) '⍎COUNT+←1'
'F1' ⎕WS 'Event' 1 2 3 '⍎COUNT+←1'      ⍝ Ditto
```

User defined Events

In addition to the standard events supported directly by Dyalog APL, you may specify your own events. For these, you **must** use event numbers; user-defined event names are not allowed.

You may use any numbers not already defined, but it is strongly recommended that you choose numbers greater than 1000 to avoid potential conflict with a future release of Dyalog APL.

You can only **generate** user-defined events under program control with $\square NQ$.

EventList   Property

Applies to   ActiveXContainer, ActiveXControl, Animation, Bitmap, BrowseBox, Button, Calendar, Circle, Clipboard, ColorButton, Combo, ComboEx, CoolBand, CoolBar, Cursor, DateTimePicker, Edit, Ellipse, FileBox, Font, Form, Grid, Group, Icon, Image, ImageList, Label, List, ListView, Locator, Marker, MDIClient, Menu, MenuBar, MenuItem, Metafile, MsgBox, OCXClass, OLEClient, OLEServer, Poly, Printer, ProgressBar, PropertyPage, PropertySheet, Rect, RichEdit, Root, Scroll, Separator, SM, Spinner, Splitter, Static, StatusBar, StatusField, SubForm, SysTrayItem, TabBar, TabBtn, TabButton, TabControl, TCPSocket, Text, Timer, TipField, ToolBar, ToolButton, ToolControl, TrackBar, TreeView, UpDown

This is a read-only property that reports the names of all the events supported by an object.

ExitApp   Event 132

Applies to   Root

If enabled, this event is reported when the user attempts to terminate a Dyalog APL/W application from the Windows Task List.

The Windows Task list displays the names of all running applications. The name displayed for a Dyalog APL/W application is defined by the Caption property of the system object *Root*. If you fail to define this property, there will be no entry for the application in the Task List.

If you wish to prevent the user from terminating your application from the Windows Task List, you may disable this event by setting its action code to ¯1. However, if you do this, your user may be puzzled as to why the operation does not work as expected. An alternative is to attach a callback function to the event which displays a message box. Not only does this allow you to provide user feedback, but you can provide confirm/cancel options. If your callback function returns a zero, your application will not be terminated.

Note that this event only provides for termination via the Windows Task List. See also the ExitWindows event.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	*'ExitApp'* or 132

ExitWindows   Event 131

Applies to   Root

If enabled, this event is reported when the user attempts to terminate the Windows Operating System. When this is done, Windows gives all running applications the opportunity to prevent it. Typically, an application that has unsaved changes will display a dialog box warning the user of this situation and offering the opportunity to cancel the termination. The default action for this event is to allow Windows to close. You can prevent this by returning a zero from a callback function. You can also prevent the user from closing Windows down by disabling the event altogether. This is achieved by setting its action code to ¯1. In most cases this is less preferable than the callback method as it does not allow you to inform the user as to why Windows won't terminate.

The event message reported as the result of $\Box DQ$, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1] Object name	character vector
[2] Event code	*'ExitWindows'* or 131
[3] Flag	0 or 1

Expanding   Event 302

Applies to   TreeView

If enabled, this event is reported by a TreeView object just before it is about to expand to show the children of the current item. This occurs when the user double-clicks the item label or clicks in the button or on the tree line to the left of the item label. The default processing for the event is to expand the tree at the corresponding point.

You may disable the expand operation by setting the action code for the event to ¯1. You may also prevent the expand from occurring by returning 0 from a callback function. You may expand a TreeView dynamically under program control by calling Expanding as a method.

The event message reported as the result of $\Box DQ$, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	*'Expanding'* or 302
[3] Item number:	Integer. The index of the item.

ExportedFns   Property

Applies to   OLEServer

ExportedFns has been superseded by the SetFnInfo method which overrides it. Use SetFnInfo instead.

This property specifies the functions to be exposed as methods by an OLEServer object.

ExportedFns may be set to 0 (none), 1 (all), or a vector of character vectors containing the names of the functions to be exported.

There are certain important restrictions concerning the type of function that you can export as a method.

Firstly, only top-level defined functions within the OLEServer may be exported; you cannot export functions in other namespaces including sub-namespaces.

Furthermore, you may not export defined operators, dynamic functions, external functions, or functions created by function assignment.

Finally, OLE does not support the concept of a dyadic function, so your exported functions must be niladic, monadic, or take an optional left argument; they may not be explicitly dyadic.

If you wish to export a new function from your OLEServer, and ExportedFns is not 1, you must explicitly reset the value of the ExportedFns property before you re-save the workspace.

ExportedVars   Property

Applies to   OLEServer

ExportedVars has been superseded by the SetVarInfo method which overrides it. Use SetVarInfo instead.

This property specifies the variables to be exposed as properties by an OLEServer object.

ExportedVars may be set to 0 (none), 1 (all), or a vector of character vectors containing the names of the variables to be exported.

Note that you may not export external variables or shared variables, or variables in other namespaces.

If you wish to export a new variable from your OLEServer, and ExportedVars is not 1, you must explicitly reset the value of the ExportedVars property before you re-save the workspace.

Expose	Event 32

Applies to   ActiveXControl, Animation, Button, Calendar, ColorButton, Combo, ComboEx, CoolBar, DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, ProgressBar, PropertyPage, RichEdit, Scroll, Spinner, Static, StatusBar, SubForm, TabBar, ToolBar, ToolControl, TrackBar, TreeView, UpDown

If enabled, this event is reported when part or all of the object's window is exposed to view. Under normal circumstances, APL repaints the exposed region automatically. However, if you have drawn unnamed graphical objects (which are **not** managed by APL) you should use this event to redraw them when required. Note that APL will itself repaint any **named** objects in the region before reporting the event.

The event message reported as the result of $\Box DQ$, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	'*Expose*' or 32
[3] Y:	y-position of top-left corner of exposed region
[4] X:	x-position of top-left corner of exposed region
[5] H:	height of exposed region
[6] W:	width of exposed region

This event cannot be disabled by setting its action code to ¯1. Similarly, setting the result of a callback function to 0 has no effect on it.

FCol   Property

Applies to   ActiveXContainer, ActiveXControl, Button, Circle, Combo, ComboEx, CoolBand, CoolBar, Edit, Ellipse, Grid, Group, Label, List, ListView, Marker, Menu, MenuItem, Poly, Rect, RichEdit, Separator, Spinner, Static, StatusBar, StatusField, TabBtn, Text, TipField, ToolBar, TreeView, UpDown

This property defines the foreground colour(s) of an object. For objects with more than one constituent part, it may specify a set of foreground colours, one for each part. A single colour is represented by a single number which refers to a standard colour, or by a 3-element vector which defines a colour explicitly in terms of its red, green and blue intensities.

If FCol is 0 (which is the default) the foreground colour is defined by your current colour scheme for the object in question. For example, if you select red as your Windows "Button Text" colour, you will by default get red writing on all your Button objects, simply by not specifying FCol or by setting it to 0.

A negative value of FCol refers to a standard Windows colour as described below. Positive values are reserved for a possible future extension.

FCol	Colour Element	FCol	Colour Element
0	Default	¯11	Active Border
¯1	Scroll Bars	¯12	Inactive Border
¯2	Desktop	¯13	Application Workspace
¯3	Active Title Bar	¯14	Highlight
¯4	Inactive Title Bar	¯15	Highlighted Text
¯5	Menu Bar	¯16	Button Face
¯6	Window Background	¯17	Button Shadow
¯7	Window Frame	¯18	Disabled Text
¯8	Menu Text	¯19	Button Text
¯9	Window Text	¯20	Inactive Title Bar Text
¯10	Active Title Bar Text	¯21	Button Highlight

If instead, FCol contains a 3-element vector, it specifies the intensity of the red, green and blue components of the colour as values in the range 0-255. For example, (255 0 0) is red and (255 255 0) is yellow.

Note that if the colour specified by FCol would normally be rendered as a dithered colour, it is instead converted to the nearest pure colour available on the device. The actual colour realised also depends upon the capabilities of the display adapter and driver, and the current Windows colour map.

For a Button, Combo, Edit, Label, List, Menu and MenuItem, FCol refers to the colour of the **text** in the object. Borders around these objects (where applicable) are drawn using the standard Windows colour. For a Static object however, FCol specifies the colour of its border.

For the Ellipse, Poly and Rect objects, FCol specifies the colour of the line drawn around the perimeter of the object. If a dashed or dotted line is used (LStyle 1-4) the "gaps" in the line are drawn using the colour specified by BCol, or are not drawn if BCol is not specified. For the Marker object, FCol specifies the colour in which the markers are drawn.

For some objects, FCol may be a vector of 3-element vectors specifying a set of colours for the constituent parts of the object. For example, a Poly object consisting of four polygons, may have a FCol property of four 3-element vectors. In addition, for graphics objects, FCol is used in place of FillCol if the latter is not specified.

FieldType   Property

Applies to   DateTimePicker, Edit, Label, Spinner

The FieldType property controls data conversion, formatting and validation . For Edit, Label and Spinner objects, FieldType controls how the Value property of these objects is interpreted.

FieldType is a character vector. If it is empty (the default) the Edit or Label object is a standard text object with no special formatting and, in the case of an Edit, no input validation. For a DateTimePicker, an empty FieldType implies the default which is *'Date'*.

For a DateTimePicker, FieldType may be one of the following:

Date	Uses Windows "short date" format
DateCentury	Uses Windows "short date" format but with a 4-digit year regardless of user preference
LongDate	Uses Windows "long date" format
Time	Uses Windows time format
Custom	Uses a special format defined by the CustomFormat property

The value of the date or time is represented by the DateTime property. Note that all validation is performed by the object itself, and it is impossible to enter an invalid value.

For an Edit, Label and Spinner, if FieldType is defined, the contents of the object are defined by its Value property (rather than by its Text property) and special formatting and validation rules are applied. FieldType may be one of the following :

Char	Character data
Numeric	Simple numeric formatting and validation
LongNumeric	Uses Windows number format
Date	Uses Windows "short date" format
LongDate	Uses Windows "long date" format
Currency	Uses Windows currency format
Time	Uses Windows time format

FieldType '*Char*' only affects an Edit object. When the user enters data into a standard single-line Edit object , the Value property is set to a number if the contents are numeric, or to a character vector if the contents do not represent a valid number. If FieldType is '*Char*', the Value property is always set to a character vector, regardless of the type of the field contents.

If FieldType is '*Numeric*', the object displays the number defined by its Value property rounded to the number of decimal places specified by its Decimals property. The decimal separator character used will be as specified by the Number format in the user's International Control Panel settings. If the object is an Edit object, the user is prevented from entering anything but a valid number. The number of decimal digits is also restricted to Decimals. When the user "leaves" the object, the number is re-formatted.

If FieldType is '*LongNumeric*', the object displays the number specified by its Value property according to the Number format in the user's International Control Panel settings. This format specifies the 1000 separator, decimal separator, decimal digits and whether or not a leading zero is inserted.

If the object is an Edit object, the user is prevented from entering anything but a valid number. However, the character specified for the 1000 separator is ignored and may be entered anywhere in the number. When the user "leaves" the object, the number is re-formatted correctly.

If the FieldType is '*Currency*', the object displays the number specified by its Value property according to the Currency format in the user's International Control Panel settings. This specifies the currency symbol and placement, the way in which a negative value is displayed, and the number of decimal places. If the object is an Edit object, the user is restricted to entering a "reasonable" value. When the user leaves the object, the number is reformatted correctly.

If the FieldType is '*Date*', the Value property represents the number of days since January 1st 1900 and is displayed using the "short date" format specified by the user's International Control Panel settings. If the object is an Edit object, the user is restricted to entering a "reasonable" date. The object will accept any numeric triplet separated by slash(/), colon (:) or space characters but checks that the day number and month number lie in the range 1-31 and 1-12 respectively and will not allow the user to enter a digit that would invalidate this. (Note that the position within the triplet of the day, month and year are as specified by the Windows short date format). However, the user is not prevented from entering an invalid date such as 31st September.

If the FieldType is '*LongDate*', the Value property represents the number of days since January 1st 1900 and is displayed using the "long date" format specified by the user's International Control Panel settings. If the object is an Edit object, its appearance and behaviour automatically switches to FieldType '*Date*' when it has the input focus and back again when it loses the focus. This allows the user to edit or input a date in a more convenient form.

For a discussion about the interpretation of 2-digit years in date fields, see the description of the YY_WINDOW parameter in User Guide.

If the FieldType is '*Time*', the Value property represents the number of seconds since midnight and is displayed using the time format specified by the user's International Control Panel settings.

When the user attempts to move the input focus away from the object, the contents are validated. If they cannot be converted to a valid number, date, or time, the object generates a BadValue event, or, if the object is associated with a Grid, the Grid (and not the Edit object) generates a CellError event. See the descriptions of these events for further details.

Note that for Edit, Label and Spinner objects, FieldType may only be specified when you create an object using ⎕*WC*.

File   Property

Applies to   Animation, Bitmap, Cursor, FileBox, Icon, Metafile, RichEdit

For an Animation, Bitmap, Cursor or Icon object, this property is either a simple character vector or a 2-element nested vector.

If it is simple, File specifies the name of the associated bitmap (.BMP), icon (.ICO) or cursor (.CUR) file.

If it is nested, the first element specifies the name of a DLL or EXE (Icon only) and the second element identifies the particular bitmap, icon or cursor in that file. The identifier may be its *name* (a character string), its *resource id* (a non-zero positive integer) or (Icon only), its *index* (0 or negative integer) within the file. As a special case, if the name of the file is an empty vector, the object is loaded from DYALOG.EXE or DYARESxx.DLL. In this case, the identifier must be a name or resource id; indices are not supported.

For a Metafile or RichEdit object, File must be simple and specifies the name of a metafile (.WMF) or Rich Text Format (RTF) file as appropriate.

When applied to a FileBox object, File contains the name of the selected file or file(s).

FileBox   Object

Purpose	Prompts user to select a file.
Parents	ActiveXControl, CoolBand, Form, Grid, OLEServer, PropertyPage, PropertySheet, Root, StatusBar, SubForm, TCPSocket, ToolBar, ToolControl
Children	Timer
Properties	Type, Caption, Directory, Filters, File, FileMode, Style, Event, Index, Data, EdgeStyle, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, FileBoxCancel, FileBoxOK, Select
Methods	Detach, Wait

The FileBox object implements the standard Windows File Selection Dialog Box. This is a "modal" object. When you create a FileBox with $\Box WC$, it is initially invisible and the user cannot interact with it. To use it, you must execute $\Box DQ$ with the name of the FileBox as its right argument. This causes the FileBox to be displayed.

During the "local" $\Box DQ$ the user may interact **only** with the FileBox, or with other applications. When the user terminates the operation (by pressing the "OK" or "Cancel" Buttons, or by closing the window) the "local" $\Box DQ$ terminates, and the FileBox disappears.

When the "local" $\Box DQ$ is terminated, the FileBox generates either an FileBoxOK(71) or FileBoxCancel(72) event. The former is generated when the user presses the "OK" button or closes the FileBox; the latter when the user presses the "Cancel" button. In both cases, the full pathname of the currently selected file is returned as the second element of the event message.

The Caption property determines the text that appears in the title bar of the FileBox window. If undefined, Caption defaults to "Save As" if FileMode is *'Write'* or to "Open" if FileMode is *'Read'*. The Directory property contains a simple character vector which specifies the initial directory from which a list of suitable files is displayed.
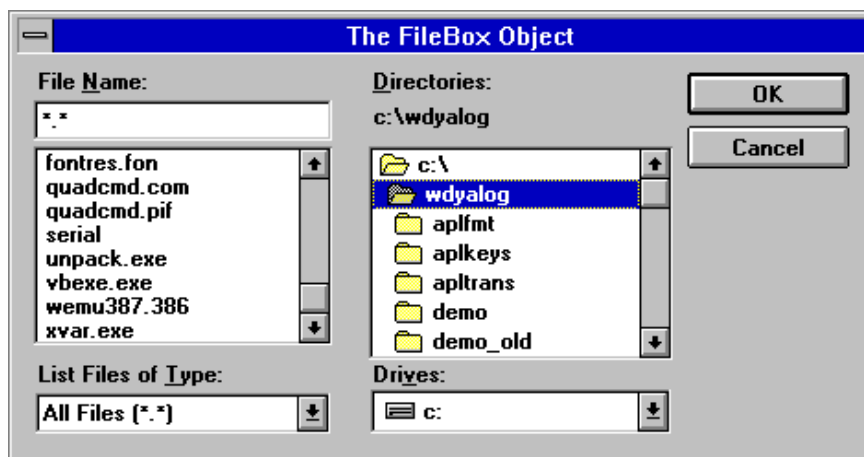
```
'F' □WC 'FileBox' 'The FileBox Object' 'C:\WDYALOG'
□DQ 'F'
```

The Filters property is a nested scalar or vector containing a list of filters.

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

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

Note that when `□DQ` terminates with FileBoxOK, the File, Directory, and Index properties are updated to reflect the contents of the fields within the FileBox.

FileBoxCancel   Event 72

Applies to   BrowseBox, FileBox

If enabled, this event is reported when a FileBox is closed because the user has pressed the "Cancel" button or closed it.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	*'FileBoxCancel'* or 72
[3] File name:	character vector containing the name of the currently selected file (empty if none)

FileBoxOK   Event 71

Applies to   BrowseBox, FileBox

If enabled, this event is reported when a FileBox is closed because the user has pressed the "OK" button.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	*'FileBoxOK'* or 71
[3] File name:	character vector containing the name of the currently selected file (empty if none)

FileMode   Property

Applies to   FileBox

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

FileRead   Method 90

Applies to   Bitmap, Cursor, Icon, Metafile, RichEdit

This method causes the object to be recreated from the file named in its File property.

The FileRead method is niladic.

If you attach a callback function to this event and have it return a value of 0, the object will not be recreated from file.

FileWrite   Method 91

Applies to   Bitmap, Cursor, Icon, Metafile, RichEdit

This method causes the object to be written to the file named in its File property.

The FileWrite method is niladic.

If you attach a callback function to this event and have it return a value of 0, the object will not be written to file. You could use this to avoid overwriting an existing file.

FillCol   Property

Applies to   Circle, Ellipse, Poly, Rect

This property defines the fill colour in a graphics object.

If FStyle is 0 (solid fill) FillCol defines the colour with which the object is filled. If FStyle is in the range 1-6 (pattern fill) it defines the colour of the lines that make up the pattern. The areas between the lines are filled using the colour specified by BCol, or are left undrawn (transparent) if BCol is not specified. If FStyle contains the name of a Bitmap object, the value of FillCol is ignored.

A single colour is represented by a single number which refers to a standard colour, or by a 3-element vector which defines a colour explicitly in terms of its red, green and blue intensities. A negative value of FillCol refers to a standard Windows colour as described below. Positive values are reserved for a possible future extension.

FillCol	Colour Element	FillCol	Colour Element
0	Default	¯11	Active Border
¯1	Scroll Bars	¯12	Inactive Border
¯2	Desktop	¯13	Application Workspace
¯3	Active Title Bar	¯14	Highlight
¯4	Inactive Title Bar	¯15	Highlighted Text
¯5	Menu Bar	¯16	Button Face
¯6	Window Background	¯17	Button Shadow
¯7	Window Frame	¯18	Disabled Text
¯8	Menu Text	¯19	Button Text
¯9	Window Text	¯20	Inactive Title Bar Text
¯10	Active Title Bar Text	¯21	Button Highlight

If instead, FillCol contains a 3-element vector, it specifies the intensity of the red, green and blue components of the colour as values in the range 0-255. For example, (255 0 0) is red and (255 255 0) is yellow. Note that the colour realised depends upon the capabilities of the display adapter and driver, and the current Windows colour map.

FillCol may also be a vector of 3-element vectors specifying a set of colours for the constituent parts of the object. For example, a Poly object consisting of four polygons, may have a FillCol property of four 3-element vectors.

Filters   Property

Applies to   FileBox

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

FirstDay   Property

Applies to   Calendar

The FirstDay property specifies the day that is considered to be the first day of the week and which appears first in the Calendar.

FirstDay is an integer whose value is in the range 0-6. The default value for FirstDay depends upon your International Settings, but in most countries is 0 meaning Monday.

Fixed   Property

Applies to   Font

This property specifies whether or not a font represented by a Font object is fixed-width or proportional. It is either 0 (fixed-width) or 1 (proportional). There is no default; the value of this property reflects the characteristic of the selected font.

FixedOrder Property

Applies to CoolBar

The FixedOrder property specifies whether or not the CoolBar displays CoolBands in the same order.

FixedOrder is a single number with the value 0 (user may re-order bands) or 1 (user may not re-order bands); the default is 0.

If FixedOrder is 1, the user may move bands to different rows, but the band order is static.

FlatSeparators Property

Applies to TabControl

The FlatSeparators property specifies whether or not separators are drawn between buttons in a TabControl object. FlatSeparators only affects a TabControl if Style is *'FlatButtons'* and is otherwise ignored.

FlatSeparators is a single number with the value 0 (no separators) or 1 (separators); the default is 0.

Flush Method 135

Applies to Root

This method forces any objects that have been created but not yet shown to be displayed. Normally, Dyalog APL/W buffers the display of new objects unless they are being created by a callback function. This event can be used to override the buffering.

The Flush method is niladic.

Font   Object

Purpose	Loads a font resource
Parents	ActiveXControl, Animation, Bitmap, Button, Calendar, Combo, ComboEx, CoolBand, DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, Metafile, OLEServer, Printer, ProgressBar, PropertyPage, PropertySheet, RichEdit, Root, Scroll, Spinner, Static, StatusBar, SubForm, TabBar, TCPSocket, TipField, ToolBar, ToolControl, TrackBar, TreeView, UpDown
Children	Timer
Properties	Type, PName, Size, Fixed, Italic, Underline, Weight, Rotate, Data, Handle, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, FontCancel, FontOK, Select
Methods	ChooseFont, Detach

This object loads a Windows font into memory ready for use by another object. The characteristics of the font are specified by its properties as follows :

PName:	A character vector containing the name of the font face, e.g.`'COURIER'`. Note that case is ignored when you specify the name, although it will be returned correctly by `⎕WG`. If you specify an empty vector, you will get an appropriate default font supplied by Windows.
Size:	An integer that specifies the character height of the font in pixels.
Fixed:	A Boolean value that specifies whether the font is fixed-width (1) or proportional (0).
Italic:	A Boolean value that specifies whether the font is italicised (1) or not (0).
Underline:	A Boolean value that specifies whether the font is underlined (1) or not (0).
Weight:	An integer in the range 0-1000 that specifies how bold or heavy the font is (1000 = most bold).
Rotate:	A numeric scalar that specifies the angle of rotation of the font in radians. The angle is measure from the x-axis in a counter-clockwise direction.

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

The values of the above properties after $\square WC$ or $\square WS$ reflect the attributes of the font which has been allocated by Windows, and not necessarily the values you have specified. Furthermore, it is possible that changing the value of one property will cause the values of others to be changed.

FontCancel   Event 242

Applies to   ActiveXControl, Button, Calendar, Combo, ComboEx, DateTimePicker, Edit, Font, Form, Grid, Group, Label, List, ListView, PropertyPage, PropertySheet, RichEdit, Root, Spinner, Static, StatusBar, SubForm, TabBtn, Text, TipField, TreeView

If enabled, this event is reported when the user has pressed the *Cancel* button or closed the font selection dialog box that is displayed by the ChooseFont method.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1] Object: ref or character vector

[2] Event code: *'FontCancel'* or 242

FontList   Property

Applies to   Printer, Root

The FontList property is a read-only property (you cannot set its value) that provides a list of available fonts.

Its value is a vector (1 per font) of 6-element character vectors, each of which is as follows :

[1]	Face name (character vector)
[2]	Character height in "points" (integer)
[3]	Fixed width or not (Boolean)
[4]	Italic or not (Boolean)
[5]	Underlined or not (Boolean)
[6]	Weight (integer)
[7]	Angle of rotation (number)

Example

```
      ↑'.' □WG 'FontList'
System                      16 0 0 0 700 0
Fixedsys                    15 1 0 0 400 0
Terminal                    12 1 0 0 400 0
MS Serif                    13 0 0 0 400 0
MS Sans Serif               13 0 0 0 400 0
Courier                     13 1 0 0 400 0
Symbol                      13 0 0 0 400 0
Small Fonts                  3 0 0 0 400 0
Dyalog Alt                  16 1 0 0 400 0
Dyalog Std                  16 1 0 0 400 0
```

Note that the list of fonts obtained from FontList for a Printer object will include TrueType fonts and printer fonts but will exclude screen fonts. FontList for Root will include TrueType fonts and screen fonts, but exclude printer-only fonts. The two lists will therefore (typically) be different.

FontObj   Property

Applies to   ActiveXContainer, ActiveXControl, Bitmap, Button, Calendar, Combo, ComboEx, DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView, Menu, MenuBar, MenuItem, Printer, PropertyPage, PropertySheet, RichEdit, Root, Spinner, Static, StatusBar, StatusField, SubForm, TabBar, TabBtn, TabControl, Text, TipField, ToolBar, ToolControl, TrackBar, TreeView

The FontObj property associates a font with an object. It specifies either the name of a Font object created by ⎕WC, or it specifies the attributes of the font directly. The use of the FontObj property to specify a font directly is supported only for compatibility with previous releases of Dyalog APL/W and may be removed in the future. The Font object provides a more efficient mechanism for managing fonts and allows greater flexibility for drawing and printing graphical text. It is recommended that all fonts be specified using Font objects and not loaded directly using the FontObj property.

If FontObj specifies the name of a Font object created by ⎕WC, it is a simple character vector.

If unspecified, the default value for FontObj is an empty character vector. For most objects, this setting implies that the font used in the object is **inherited** from its parent object. However, CoolBar, Menu, MenuBar, StatusBar, TipField, ToolBar, and ToolControl objects do not inherit their font.

Note that the default value of FontObj for Root is also an empty character vector and that this implies the Windows default GUI font, which is a Windows user preference setting.

Note however that it is not currently possible to specify the font for Menu and MenuItem objects which are the direct descendants of a MenuBar. Nor is it possible to specify the font used for the Caption in a Form.

If FontObj specifies a font directly, it may be either an empty character vector (this is its default value) or as an array containing up to 7 elements as follows :

[1]	Face name of requested font (character vector)
[2]	Character height in pixels (integer)
[3]	Fixed width or not (Boolean)
[4]	Italic or not (Boolean)
[5]	Underlined or not (Boolean)
[6]	Weight (integer)
[7]	Angle of rotation (integer)

When you assign a value to the FontObj property of an object, Windows actually supplies the font that most closely matches the attributes you have specified. The matching rules it uses are complex, and may be found in the appropriate Windows documentation. The value of the FontObj property reflects the attributes of the font which has been allocated by Windows, not the value you originally specified.

A list of available fonts and their attributes may be obtained from the FontList property of the Root object ". ".

The "Face name" is the name assigned to the font by Windows. The face name of the standard Dyalog APL screen font is *'Dyalog Std'*. Note that case is ignored when you specify the name, although it will be returned correctly by ⎕*WG*.

The size of the font is specified in terms of its height in pixels. If Windows cannot supply exactly the size you request, it will supply the nearest below that.

A value of 1 in the third element requests a fixed-width font, as opposed to a proportional one. This attribute is given the maximum weighting by Windows in choosing a matching font. A value of 1 in the fourth and fifth elements requests the font attributes **italic** and **underlined** respectively. Windows will add these attributes to an existing font if they don't physically exist. For example, you **can** obtain italic and underlined APL characters from the standard APL font. The weight is a number in the range 0 to 1000 which specifies how **feint** or **bold** the characters appear. The larger the number, the bolder the font. The angle of rotation is measured in 1/10ths of a degree from the x-axis in a counter-clockwise direction. Its default value is 0.

FontOK   Event 241

Applies to   ActiveXControl, Button, Calendar, Combo, ComboEx,
                        DateTimePicker, Edit, Font, Form, Grid, Group, Label, List,
                        ListView, PropertyPage, PropertySheet, RichEdit, Root, Spinner,
                        Static, StatusBar, SubForm, TabBtn, Text, TipField, TreeView

If enabled, this event is reported when the user has pressed the *OK* button in the font
selection dialog box that is displayed by the ChooseFont method.

The event message reported as the result of *⎕DQ*, or supplied as the right argument to
your callback function, is a 4-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	*'FontOK'* or 241
[3] Font specification	nested vector
[4] Colour	RGB triplet

The font specification in the 3rd element of the event message is a 7-element nested
vector that describes the chosen font. See Font Object for further details.

The colour specification in the 4th element of the event message is a 3-element integer
vector of RGB values for the colour chosen by the user.

Form   Object

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

Parents   ActiveXControl, Form, OLEClient, OLEServer, Root, SubForm, TCPSocket

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

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

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

Methods   Animate, ChooseFont, Detach, GetFocus, GetTextSize, ShowSIP, Wait

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

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

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

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

If Sizeable is 1, the window will have a double-line border, regardless of the values of other properties. If Sizeable is 0, and any one or more of Moveable, SysMenu, MaxButton, MinButton or Border is 1, the window will have a 1-pixel border. Only if all these properties are 0 will the window be without borders.

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

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

```
'F' ⎕WC 'Form' '' ('EdgeStyle' 'Dialog')('Border' 2)
```

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

Under Pocket APL, the appearance, behaviour and the default values of certain properties of a Form is different.

- A Pocket APL Form cannot be minimised.
- By default, SysMenu, Sizeable and Moveable are 0 and State is 2. This creates a full-screen Form which the user cannot move or resize. The title bar of the Form coincides with the *Start* title bar which has a circular button in the top right corner. This will be labelled *X* or *OK*, depending upon the value of the OKButton property. Note that the *X* button merely hides the Form and does not close it (see OKButton for further details).
- If you create a Form with SysMenu 1, you get an independent window with its own title bar and its own separate *X* button that may be used to close it.
- The values of the MinButton, MaxButton and HelpButton properties have no direct effect, although setting any of these properties to 1 causes SysMenu to be 1.

The VScroll and HScroll properties determine whether or not a Form has a vertical and horizontal scrollbar respectively. These properties are set to ¯1 to obtain a scrollbar. Their default value is 0 (no scrollbar). The Range property is a 2-element vector that specifies the maximum value for the vertical and horizontal scrollbars respectively. The Step property is a 4-element vector that specifies the sizes of the small and large change. Its first two elements refer to the vertical scrollbar, elements 3 and 4 refer to the horizontal scrollbar. The Thumb property is a 2-element vector that both reports and sets the position of the thumb in the vertical and horizontal scrollbars respectively. When the user attempts to move the thumb in one of the scrollbars, the Form generates a VScroll or HScroll event.

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

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

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

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

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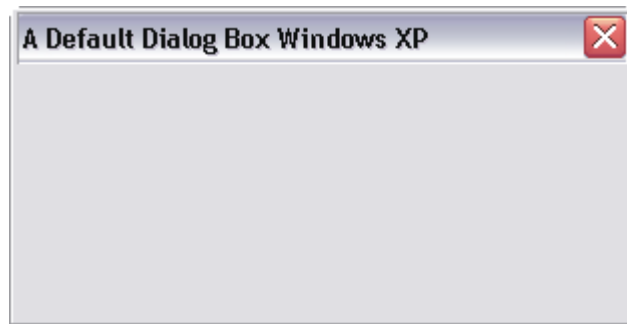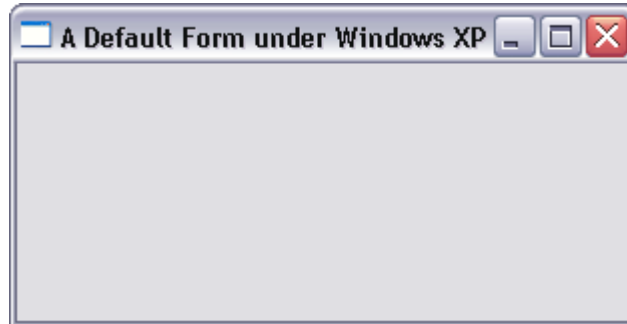
A Form can be created as a child of another Form. If so, it has the following characteristics :

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

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

Formats   Property

Applies to   Clipboard

This is a "read-only" property that identifies the formats in which data is currently available in the clipboard. It is a vector of character vectors containing the names of the corresponding Clipboard properties for which data may be obtained using `⎕WG`.

In the following example data was copied to the Windows clipboard from Microsoft Excel.

```
      'CL' ⎕WC 'Clipboard'

      CL.Formats
Metafile  Bits  CBits  Text  RTFText
```

FormatString   Property

Applies to   Edit, Grid, Label, Spinner
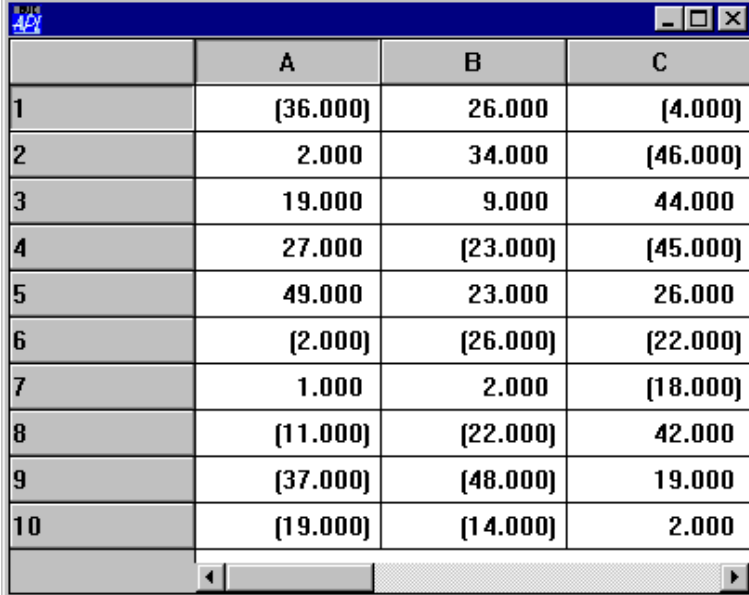
The FormatString property specifies one or more `⎕FMT` format specifications to be used to format the Value property in an object with FieldType `'Numeric'`, or the Values property in a Grid. In the latter case, it is either a simple character vector that specifies the format specification for the entire Grid, or a vector of character vectors. If it is a vector, its elements are mapped to individual cells via the CellTypes property. When applied to any other object, FormatString must be a simple character vector.

The interpreter derives the text to be displayed in a cell by calling `⎕FMT` with a left argument of the corresponding element of FormatString and a right argument of the cell value. If the format specification is invalid, the text displayed is blank.

When a formatted Edit object receives the focus, it redisplays the contents in its raw (unformatted) form. When the Edit loses the focus, its contents are reformatted. When the user moves to a formatted Grid cell, the text remains formatted until the user presses a non-movement key or enters *in-cell* mode. The data is then redisplayed in its raw form for editing. Data in the cell is reformatted when the user moves away.

In a Grid, formatted data may be aligned vertically using the AlignChar property as illustrated in the following example.

```
'F'⎕WC'Form'
'F.G'⎕WC'Grid'(¯50+?10 10ρ100)(0 0)(100 100)
'F.G'⎕WS'FormatString' 'M<(>N<)>F12.3'
'F.G'⎕WS'AlignChar' '.'
```

	A	B	C
1	(36.000)	26.000	(4.000)
2	2.000	34.000	(46.000)
3	19.000	9.000	44.000
4	27.000	(23.000)	(45.000)
5	49.000	23.000	26.000
6	(2.000)	(26.000)	(22.000)
7	1.000	2.000	(18.000)
8	(11.000)	(22.000)	42.000
9	(37.000)	(48.000)	19.000
10	(19.000)	(14.000)	2.000

FrameContextMenu   Event 411

Applies to   Form, SubForm

If enabled, this event is reported when the user clicks and releases the right mouse button over the non-client area of an object, e.g. the title bar in a Form.

The event message reported as the result of ⎕DQ, or supplied as the right argument to your callback function, is a 2-element vector as follows :

The event message reported as the result of ⎕DQ, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	*'FrameContextMenu'* or 411
[3] Y:	y-position of the mouse (number)
[4] Y:	x-position of the mouse (number)

For further details, see ContextMenu Event.

FStyle Property

Applies to Circle, Ellipse, Poly, Rect

This property determines how a graphics object is filled. It takes one of the following values, or, if the object has more than one component, a vector of such values.

FStyle	Effect
‾1	hollow (no fill). This is the default.
0	solid fill
1	hatch fill with horizontal lines
2	hatch fill with vertical lines
3	hatch fill with diagonal lines at 135 degrees
4	hatch fill with diagonal lines at 45 degrees
5	hatch fill with horizontal and vertical lines
6	hatch fill with criss-crossing diagonal lines
str	the name of a Bitmap object which is used to fill the object.

For example, to fill an object with criss-crossing diagonal lines you would specify (`'FStyle'` 6). If the object contained two components, you could fill the first one with criss-crossing diagonal lines, and the second one with a Bitmap called `'YES'`, with the specification (`'FStyle'` 6 `'YES'`)

Note: If the size of the Bitmap is 8x8 APL uses a Windows "brush" to fill the object. If not, it uses "tiling". Filling with a brush is faster.

FullRowSelect   Property

Applies to   ListView, TreeView

Requires the Windows Custom Control Library COMCTL32.DLL, Version 4.70 or higher.

The FullRowSelect property specifies whether or not the entire row is highlighted when an item in a ListView or a TreeView is selected.

FullRowSelect is a single number with the value 0 (only the item name is highlighted) or 1 (the whole row is highlighted); the default is 0.

For a ListView, FullRowSelect only applies if its View property is set to *'Report'*.

The picture below illustrates the effect on the appearance of a ListView object, of setting FullRowSelect to 1. Note that, in this example, CheckBoxes is also set to 1.
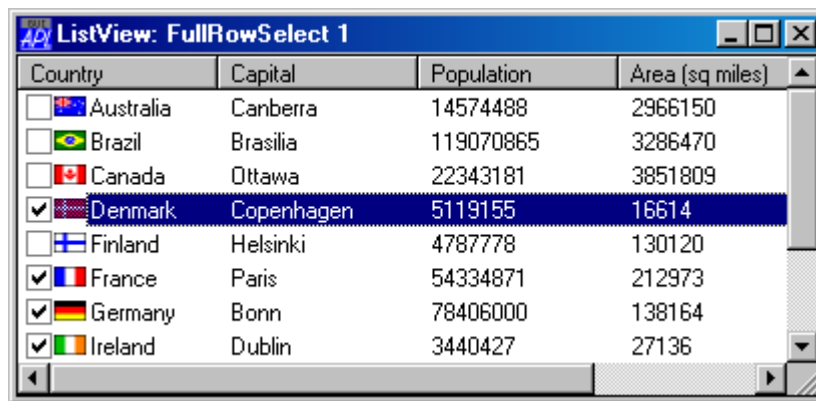
GetBuildID   Method 192

Applies to   Root

This method is used to obtain the Build ID of a Dyalog APL executable.

The argument to GetBuildID is θ or a single item as follows:

> [1] File name: character vector

The (shy) result is an 8-element character vector of hexadecimal digits that represents the Build ID.

If the argument is θ, the build id is that of the current version of Dyalog APL that is running the expression.

Note that although this method is designed to uniquely identify different versions of Dyalog APL by its check-sum, it may be used to obtain a check-sum for *any* arbitrary file.

Examples

```
      GetBuildID θ
38091b76
      GetBuildID 'E:\DYALOG81\DYALOG.EXE'
cbf0d376
      GetBuildID 'C:\AUTOEXEC.BAT'
4a29334d
```

Note that if the file does not exist, the result is 00000000.

GetCellRect Method 201

Applies to Grid

This method returns the rectangle associated with a particular cell in a Grid.

The argument to GetCellRect is a 2-element vector as follows:

[1] Row: integer

[2] Column: integer

The result is a 2-element nested vector. The first element contains the y and x-coordinate of the top-left corner of the cell. The second element contains the height and width of the cell.

The result is reported in terms of the coordinate system of the Grid object.

GetCommandLine Method 145

Applies to Root

The GetCommandLine method returns the command line that was used to start the current Dyalog APL session or application.

The GetCommandLine method is niladic.

The result is a character vector. For example:

```
      GetCommandLine
"C:\Dyalog10\dyalog.exe" -Dw YY_WINDOW=-30

      □←2 □NQ '.' 'GetCommandLine'
"C:\Dyalog10\dyalog.exe" -Dw YY_WINDOW=-30
```

GetCommandLineArgs   Method 148

Applies to   Root

The GetCommandLineArgs method returns the command and the arguments to the command that was used to start the current Dyalog APL session or application.

The GetCommandLineArgs method is niladic.

The result is a vector of character vectors. For example:

```
      GetCommandLineArgs
 C:\Dyalog10\dyalog.exe  -Dw  YY_WINDOW=-30

      DISPLAY 2 ⎕NQ '.' 'GetCommandLineArgs'
.→------------------------------------------------.
| .→--------------------. .→--. .→------------. |
| |C:\Dyalog10\dyalog.exe| |-Dw| |YY_WINDOW=-30| |
| '----------------------' '---' '-------------' |
'∊------------------------------------------------'
```

GetComment   Method 222

Applies to   Grid

This method is used to retrieve the comment associated with a cell.

The argument to GetComment is a 2-element array as follows:

[1] Row:	integer
[2] Column:	integer

For example, the following expression retrieves the comment associated with the cell at row 3, column 1.

```
      F.G.GetComment 3 1
 1 3  Hello  175 100
```

If there is no comment associated with the specified cell, the result is a scalar 1.

GetDayStates   Event 266

Applies to   Calendar

If enabled, this event is reported when a Calendar object requires the APL program to provide *day state* information for the range of dates it is about to display.

The Calendar object displays day numbers using either the normal or the bold font attribute. However, it does not store this information beyond the month or months currently displayed.

When the Calendar control scrolls (and potentially at other times), it generates a GetDayStates event to ask you (the APL program) to tell it which of the dates that are about to be shown, should be displayed using the bold font attribute.

If you wish any dates to be displayed using the bold font attribute, you **must** attach a callback function to this event which returns day state information in its result.

By default, all dates are displayed using the normal font attribute, so you need only do this if you want any dates highlighted in bold.

You may not disable or nullify the operation that caused GetDayStates to fire by setting the action code for the event to ¯1 or by returning 0 from a callback function.

The event message reported as the result of ⎕DQ, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	*'GetDayStates'* or 266
[3] First Date:	an integer (IDN)
[4] Last Date:	an integer (IDN)
[5] Bold Dates:	an integer vector of IDNs.

When the callback function is invoked, the 3rd and 4th elements of the event message contain IDNs for the first and last date in the range of dates that the Calendar object is about to display. The 5th element of the event message contains those IDNs *within this range of dates* that the Calendar control already knows are to be displayed using the bold font attribute. This will typically be empty.

The result of your callback function should be the same event message with only the 5th element modified in any way. This should contain the IDNs of the dates (within the range specified by the 3rd and 4th elements) that are to be displayed using the bold font attribute.

Example

Suppose that you keep a variable *BOLD_DATES* in the Calendar object. This variable is a vector of IDN values that defines those dates that the user has somehow identified as special and that you wish to display in bold, The following callback function could be applied:

```
      ∇ MSG←DAYSTATES MSG;MASK;⎕IO
[1]    ⍝ Callback function for the GetDayStates event
[2]    ⍝ Object (⊃MSG) contains a variable BOLD_DATES that
[3]    ⍝ defines ALL IDNs that are to be displayed in bold
[4]    ⍝ We need to return only those that fall within range
[5]    ⍝ of dates that are about to be displayed by Calendar
[6]     ⎕CS⊃MSG
[7]     ⎕IO←1
[8]     MASK←BOLD_DATES≥3⊃MSG
[9]     MASK←MASK∧BOLD_DATES≤4⊃MSG
[10]    MSG[5]←⊂MASK/BOLD_DATES
      ∇
```

You may also set the font attribute for particular days by calling GetDayStates as a method.

For example, to set the bold attribute for IDN 36048 (11 September 1998) in a Calendar object called '*F.CAL1*', you could execute the expression:

```
    F.CAL1.GetDayStates 36048 36048 36048
```

To *clear* the bold attribute for the same day:

```
    F.CAL1.GetDayStates 36048 36048 ⍬
```

Note that the Calendar object will ignore any IDNs you specify that are outside the range of dates that it is currently displaying.

GetEnvironment   Method 510

Applies to   Root

This method is used to obtain information about one or more parameters that were specified in the APL command line, the Windows registry, or defined as environment variables. These parameters may be *official* Dyalog APL parameters or ones of your own invention. If a value is defined in several places (for example, MAXWS in the command line overriding MAXWS in the registry), GetEnvironment follows exactly the same logic as is used by Dyalog APL itself and so obtains the same value.

The argument to GetEnvironment is a single item as follows:

        [1] Parameter name(s):   see below

Parameter names is simple character vector or vector of character vectors specifying one or more parameters.

The result is a simple character vector or a vector of character vectors.

Examples:

```
    GetEnvironment 'DYALOG'
C:\Dyalog82

    GetEnvironment ⊂'DYALOG' 'APLNID'
 C:\Dyalog82  0
```

Note that you may use GetEnvironment to obtain the values of your own arbitrary parameters given on the APL command line, defined in the registry, or specified as environment variables.

Note that GetEnvironment is not supported by DYALOG10.DLL because it does not use parameters.

GetEventInfo   Method 551

Applies to   OCXClass, OLEClient

This method is used to obtain information about a particular event or set of events supported by a COM object.

For each event supported by a COM object, the author will have registered the data type of its result (if it has a result), a help message or description of the event (optional) and the name and data type of each of its parameters. These event parameters make up the array returned by $\square DQ$ or supplied as an argument to your callback function. The GetEventInfo method returns this information.

The argument to GetEventInfo is a single item as follows:

> [1] Event name(s):   see below

Event name(s) is a simple character vector or a vector of character vectors specifying one or more names of events supported by the object.

The result is a nested vector with one element per event name. Each element of this vector is itself a vector of 2-element character vectors. For each event, the first item describes the help message or description (if any) registered for the event and the data type of its result. Each of the remaining elements contains a parameter name and its corresponding data type.

For example,

```
      CLNAME←'Microsoft Multimedia Control, Version 6.0'
      'MM' □WC 'OCXClass' CLNAME

      MM.EventList
 Done  BackClick  PrevClick  NextClick  PlayClick  ...

      DISPLAY ↑MM.GetEventInfo 'Done'
.→--------------------------------------------------.
↓ .→----------------------------. .→------.         |
| |Occurs when an MCI command ...| |VT_VOID|         |
| '----------------------------' '-------'          |
| .→---------.                   .→--------------.  |
| |NotifyCode|                   |VT_PTR to VT_I2| |
| '----------'                   '--------------' |
'∈--------------------------------------------------'
```

Note that if the event does not produce a result, the data type of the result is reported as
'VT_VOID'.

GetFocus   Method 511

Applies to   ActiveXControl, Animation, Button, Calendar, ColorButton, Combo, ComboEx, CoolBar, DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, ProgressBar, PropertyPage, PropertySheet, RichEdit, Root, Scroll, SM, Spinner, Static, StatusBar, SubForm, TabBar, TabControl, ToolBar, ToolControl, TrackBar, TreeView, UpDown

This method is used to obtain the name of the object that currently has the input focus.

The GetFocus method is niladic.

The result is a simple character vector. An empty result indicates that no Dyalog APL GUI object has the input focus.

GetItemHandle   Method 313

Applies to   TreeView

This method is used to obtain the handle of a particular item in a TreeView object.

The argument for GetItemHandle is a single item as follows:

      [1] Item number: Integer.

Item number is the index of the item concerned.

The result is an integer containing the handle of the item.

GetItemPosition Method 323

Applies to ListView

This method is used to obtain the position of a particular item in a ListView object.

The argument for GetItemPosition is a single item as follows:

[1] Item number: Integer.

Item number is the index of the item concerned.

The result is a 2-element vector containing the position of the item.

GetItemState Method 306

Applies to ListView, TreeView

This method is used to obtain the status of a particular item in a ListView or TreeView object.

The argument for GetItemState is a single item as follows:

[1] Item number: Integer.

Item number is the index of the item concerned.

The result indicates the state of the item as the sum of one or more of the following codes:

1	Item has the focus
2	Item is selected
8	Item is highlighted for dropping
16	Item is displayed in bold text
32	Item is expanded
64	Item is or has been expanded
4096	Item is checked (see CheckBoxes)

GetMethodInfo   Method 552

Applies to   OCXClass, OLEClient

This method is used to obtain information about a particular method or set of methods supported by a COM object.

For each method supported by a COM object the author will have registered a help message or description of the method (this is in fact optional), the data type of its result (if it has a result), and the name and data type of each of the parameters that must be supplied when you invoke it. The GetMethodInfo method returns this information.

The argument to GetMethodInfo is a single item as follows:

    [1] Method name(s):   see below

Method name(s) is a simple character vector or a vector of character vectors specifying one or more names of methods supported by the object.

The result is a nested vector with one element per method name. Each element of this vector is itself a vector of 2-element character vectors. For each method, the first item describes the help message or description (if any) registered for the method and the data type of its result. Note that if the event does not produce a result, the data type of the result is reported as *'VT_VOID'*. Each of the remaining elements contains a parameter name and its corresponding data type.

For example,

```
      CLNAME←'Microsoft Multimedia Control, Version 6.0'
      'MM' ⎕WC 'OCXClass' CLNAME

      MM.MethodList
 AboutBox   Refresh   OLEDrag

      DISPLAY ↑ MM.GetMethodInfo 'AboutBox'
.→--------------.
↓ .⊖. .→------. |
| | | |VT_VOID| |
| '-' '-------' |
'∊--------------'
```

GetMinSize   Method 275

Applies to   Calendar

This method is used to obtain the minimum size that you must specify for a Calendar object for it to display a complete month.

The GetMinSize method is niladic.

The result is a 2-element numeric vector containing the minimum height and width required for the object to display a complete month.

GetParentItem   Method 312

Applies to   TreeView

This method is used to obtain the index of the parent of a particular item in a TreeView object.

The argument for GetParentItem is a single item as follows:

      [1] Item number: Integer.

Item number is the index of the item concerned.

The result is an integer containing the index of the parent item.

GetPropertyInfo   Method 550

Applies to   OCXClass, OLEClient

This method is used to obtain information about a particular property or set of properties supported by a COM object.

For each property supported by a COM object, the author will have registered the property name, its data type, and an optional help message or description of the property. GetPropertyInfo returns this information.

The argument to GetPropertyInfo is a single item as follows:

> [1] Property name(s):   see below

Property name(s) is a simple character vector or a vector of character vectors specifying one or more names of properties supported by the object.

The result is a nested vector with one element per property name. Each element of this vector is itself a 2-element vector of character vectors containing the data type and help message for the corresponding property.

For example,

```
      CLNAME←'Microsoft Multimedia Control, Version 6.0'
      'MM' □WC 'OCXClass' CLNAME


      MM.PropList
Type  DeviceType  AutoEnable  PrevVisible ...

      DISPLAY ↑MM.GetPropertyInfo 'PrevVisible'
.→---------------------------------------------------------.
↓ .→------------------------------------------. .→------. |
| |Determines if the Prev button is visible.| |VT_VOID| |
| '------------------------------------------' '-------' |
| .→-.                                          .→------. |
| |⌈P|                                          |VT_BOOL| |
| '--'                                          '-------' |
' ∈-------------------------------------------------------'
```

If the data type of a property is VT_USERDEFINED, it means that the property may assume one of a set of values defined by a Type List. In this case, the name of the type list is returned in place of the string "VT_USERDEFINED". Further information can be obtained using GetTypeInfo with this name as a parameter.

Certain properties (for example, Posn and Size) are not native properties of an OLE Control, but are added by Dyalog APL. For these properties, the data type is reported as VT_APLINTERNAL.

GetTextSize Method 92

Applies to ActiveXControl, Animation, Bitmap, Button, Calendar, ColorButton, Combo, ComboEx, CoolBar, DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, Printer, ProgressBar, PropertyPage, RichEdit, Root, Scroll, SM, Spinner, Static, StatusBar, SubForm, TabBar, TabControl, ToolBar, ToolControl, TrackBar, TreeView, UpDown

The GetTextSize method obtains the size of the bounding rectangle of a text item in a given font. The result is given in the co-ordinate system of the object in question. This method is useful for positioning Text objects.

GetTextSize duplicates the functionality of the TextSize property. It is recommended that you use GetTextSize instead of TextSize which may be removed in a future release of Dyalog APL.

The argument to GetTextSize is a 1 or 2-element array as follows:

[1] Text item: character array

[2] Font name: character vector

When you invoke GetTextSize you give the text item in whose size you are interested and, optionally, the name of a Font object. The text item may be a simple scalar, a vector or a matrix. If the Font is omitted, the result is given using the current font for the object in question.

Examples

```
      'F'⎕WC'Form'
      F.GetTextSize'Hello World'
3.385416667 10.7421875

      'FNT1' ⎕WC 'Font' 'Arial' 72
      F.GetTextSize'Hello World'  '#.FNT1'
18.75 65.4296875

      F.Coord←'Pixel'
      F.FontObj←'FNT1'
```

```
      F.GetTextSize'Hello World'
16 77
```

GetTypeInfo Method 553

Applies to   OCXClass, OLEClient

This method is used to obtain information about a Type List supported by a COM object.

The argument to GetTypeInfo is a single item as follows:

[1] Type List name(s):   see below

Type List name(s) is a simple character vector or a vector of character vectors specifying one or more names of type lists supported by the object.

The result is a nested vector with one element per Type List. Each element of this vector is itself a 3-element vector of character vectors made up as follows:

[1] Name of Constant: character vector

[2] Value: (usually) numeric

[3] Description: character vector

GetVisibleRange   Method 262

Applies to   Calendar

This method is used to obtain the range of dates that is currently visible in a Calendar object.

The GetVisibleRange method is niladic.

The result is a 2-element integer vector containing the first and last dates currently displayed by the object, reported as IDNs.

GotFocus   Event 40

Applies to   ActiveXControl, Animation, Button, Calendar, ColorButton, Combo, ComboEx, DateTimePicker, Edit, Form, Grid, Group, List, ListView, MDIClient, ProgressBar, PropertyPage, RichEdit, Scroll, Spinner, SubForm, TrackBar, TreeView

If enabled, this event is generated when the user has moved the keyboard focus to a new object by clicking the left mouse button, pressing TAB, or using a cursor key.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1] Object:	ref or character vector (object that has received the focus)
[2] Event code:	*'GotFocus'* or 40
[3] Object:	ref or character vector (object which previously had the focus)

The third element (object name) is empty if the focus was obtained from another application window.

The GotFocus event is generated **after** the focus has changed. The default processing is therefore to take no action. However, if you inhibit the event by returning a 0 from your callback function, the focus is automatically restored to the object (or external application) that had lost it.

GreetBitmap   Method 138

Applies to   Root

This method is used to display or remove a bitmap, typically during initialisation of a Dyalog APL runtime application.

The argument to GreetBitmap is `0` or a 2 element vector as follows:

> [1] Display: `0` = off, `1` = on.
>
> [2] Bitmap file name: Character vector.

If the argument is `0`, the bitmap is removed.

The image may also be displayed initially by setting parameter: **greet_bitmap** on the command line, e.g.:

```
c:\myapp\dyalogrt greet_bitmap=mylogo myws
```

The image is displayed until either an untrapped error occurs, causing the interpreter to (attempt to) display the session window, or the GreetBitmap method is called.

Grid   Object

Purpose	This object displays data in a spreadsheet format and allows the user to change it.
Parents	ActiveXControl, CoolBand, Form, Group, PropertyPage, SubForm
Children	Bitmap, BrowseBox, Button, Circle, ColorButton, Combo, Cursor, Edit, Ellipse, FileBox, Font, Icon, Image, Label, Marker, Menu, MsgBox, OCXClass, Poly, Rect, Spinner, Text, Timer, TrackBar
Properties	Type, Values, Posn, Size, FCol, BCol, Coord, Border, Active, Visible, Event, VScroll, HScroll, SelItems, Sizeable, Dragable, FontObj, CursorObj, AutoConf, Index, YRange, XRange, Data, Attach, TextSize, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, FormatString, RowTitles, ColTitles, CurCell, TitleWidth, CellHeights, CellWidths, TitleHeight, CellFonts, Input, CellTypes, AutoExpand, CellSelect, ResizeRows, ResizeCols, ResizeRowTitles, ResizeColTitles, ClipCells, InputModeKey, InputMode, GridFCol, GridBCol, ShowInput, CellSet, RowTitleFCol, ColTitleFCol, RowTitleDepth, ColTitleDepth, RowTitleAlign, ColTitleAlign, OverflowChar, AlignChar, GridLineFCol, GridLineWidth, RowLineTypes, ColLineTypes, EnterReadOnlyCells, RowTitleBCol, ColTitleBCol, Translate, Accelerator, AcceptFiles, KeepOnClose, Redraw, InputProperties, MethodList, ChildList, EventList, PropList
Events	AddCol, AddRow, CellChange, CellChanged, CellDblClick, CellDown, CellError, CellMove, CellUp, ClickComment, Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, FontCancel, FontOK, GotFocus, GridCopy, GridCopyError, GridCut, GridDelete, GridDropSel, GridKeyPress, GridPaste, GridPasteError, GridSelect, Help, HideComment, IndexChanged, KeyPress, LostFocus, MouseEnter, MouseLeave, Select, SetColSize, SetRowSize, ShowComment
Methods	AddComment, Animate, CellFromPoint, ChooseFont, ColChange, DelCol, DelComment, DelRow, Detach, DuplicateColumn, DuplicateRow, GetCellRect, GetComment, GetFocus, GetTextSize, LockColumns, LockRows, RowChange, SetCellSet, SetCellType, ShowSIP, Undo

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

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

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

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

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

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

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

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

If the Input property is empty (the default) the user may browse the data in the spreadsheet but may not alter it. Furthermore, no feedback is provided as to which is the current cell. If the Input property specifies the name of an object that is the child of the Grid itself, this object *floats* from cell to cell as the user moves around the spreadsheet, and the current cell is identified by its presence. If the Input property specifies the name of an external object (that is, an object that is *not* a child of the Grid), the contents of the current cell are copied into that object as the user moves around the spreadsheet. In addition, the current cell is identified by a thick border. In either case, the associated object is used to impose formatting and validation.

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

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

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

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

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

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

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

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

The Grid object maintains a buffer of the most recent 8 changes made by the user since the Values property was last set by $\square WC$ or $\square WS$. Your application can restore these changes one by one using the Undo method. The Undo method restores the most recent change made by the user and removes that change from the undo stack. It is therefore not possible to "undo an undo".

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

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

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

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

If you move the mouse pointer over any of the four edges of a selected block of cells, the cursor changes to an arrow. You may now click and drag the border of the selected cells with the mouse. If you press the Ctrl key at the same time, the contents of the selected cells are *copied* to the new location, replacing the values in the block of cells onto which they are dropped. Otherwise, the operation is treated as a *move* and the original block of cells is emptied. This operation also generates a GridDropSel event. You may only move or copy a *single* block of cells in this way.

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

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

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

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

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

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

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

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

The HScroll and VScroll properties specify whether or not horizontal and vertical scrollbars are displayed. Either property may be given the value ¯3 which forces the corresponding scrollbar to appear always.

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

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

You may lock individual rows and columns using the LockRows and LockColumns methods.

GridBCol   Property

Applies to   Grid

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

GridBCol may be a 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or it may be a scalar that defines a standard Windows colour element (see BCol for details). Its default value is 0 which obtains the colour defined for Window Background.

GridCopy   Event 191

Applies to   Grid

If enabled, this event is reported when the user presses Ctrl+Insert and there are selected cells in the Grid. The default action of the event is to copy the contents of the selected block(s) of cells to the clipboard. You may disable this effect entirely by setting the action code of the event to ¯1. You may also disable the copy operation by returning 0 from a callback function.

The event message reported as the result of $\Box DQ$, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	`'GridCopy'` or 191
[3] Start:	2-element integer vector or matrix containing the row, column address(es) of the top left cell(s) in the selected block(s)
[4] End:	2-element integer vector or matrix containing the row, column address(es) of the bottom right cell(s) in the selected block(s)
[5] Data:	2-element nested vector. The first element is a matrix containing the values of the selected block(s) of cells. This is the data that will be copied to the clipboard. The second element is a Boolean matrix containing the values of the CellSet property for the selected block of cells.

Note that the values of Start and End are sensitive to the index origin, $\Box IO$.

If more than one block of cells is selected, Start and End are matrices whose rows identify the start and end cells of each of the selected blocks, and Data is the contents of the selected blocks catenated along the appropriate dimension according to their relative positions in the Grid.

You may copy cells under program control by calling GridCopy as a method.

To copy a specific block of cells to the clipboard whether or not they are selected, you must specify the Start and End parameters. For example, the following expression will copy the 3x3 block of cells in the top-left of the Grid ($\Box IO$ is 1) to the clipboard:

```
gridname.GridCopy (1 1) (3 3)
```

If you omit these parameters, the currently selected block of cells will be copied to the clipboard. If no cells are selected, the entire contents of the Grid will be copied. i.e.

```
gridname.GridCopy θ
```

The data copied to the clipboard is registered in Dyalog (APL internal),Wk3 (Lotus), XlTable (Excel) and tab/new-line delimited text formats.

GridCut Event 190

Applies to Grid

If enabled, this event is reported when the user presses Shift+Delete and there are selected cells in the Grid. The default action of the event is to copy the contents of the selected block(s) of cells to the clipboard and then to empty the selected cells. You may disable this effect entirely by setting the action code of the event to ¯1. You may also disable the cut operation by returning 0 from a callback function.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	*'GridCut'* or 190
[3] Start:	2-element integer vector or matrix containing the row, column address(es) of the top left cell(s) in the selected block(s)
[4] End:	2-element integer vector or matrix containing the row, column address(es) of the bottom right cell in the selected block(s)
[5] Data:	2-element nested vector. The first element is a matrix containing the values of the selected block(s) of cells. This is the data that will be copied to the clipboard. The second element is a Boolean matrix containing the values of the CellSet property for the selected block of cells.

Note that the values of Start and End are sensitive to the index origin, $\square IO$.

If more than one block of cells is selected, Start and End are matrices whose rows identify the start and end cells of each of the selected blocks, and Data is the contents of the selected blocks catenated along the appropriate dimension according to their relative positions in the Grid.

The data copied to the clipboard is registered in Dyalog (APL internal),Wk3 (Lotus), XlTable (Excel) and tab/new-line delimited text formats.

GridDelete   Event 193

Applies to   Grid

If enabled, this event is reported when the user presses Delete and there are selected cells in the Grid. The default action of the event is to empty the selected cells. You may disable this effect entirely by setting the action code of the event to ¯1. You may also disable the delete operation by returning 0 from a callback function.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	*'GridDelete'* or 193
[3] Start:	2-element integer vector or matrix containing the row, column address(es) of the top left cell(s) in the selected block(s)
[4] End:	2-element integer vector or matrix containing the row, column address(es) of the bottom right cell in the selected block(s)

Note that the values of Start and End are sensitive to the index origin, $\square IO$.

If more than one block of cells is selected, Start and End are matrices whose rows identify the start and end cells of each of the selected blocks.

GridDropSel   Event 195

Applies to   Grid

If enabled, this event is reported when the user drag/drops a selected block of cells in the Grid. The default action is that the contents of the selected cells replace the values in the block of cells onto which they are dropped and this block now becomes selected.
You may disable the drag/drop facility entirely by setting the action code of the event to ¯1. You may also disable an individual drag/drop operation by returning 0 from a callback function.

The event message reported as the result of *⎕DQ*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	*'GridDropSel'* or 195
[3] Start:	2-element integer vector containing the row, column address of the top left cell in the selected block
[4] Size:	2-element integer vector containing the number of rows and columns in the selected block
[5] Target:	2-element integer vector containing the row/column address of the top left cell onto which the selected block is being dropped

[6] Shift State:	sum of shift key codes (number)
	    1 = Shift key is down
	    2 = Ctrl key is down
	    4 = Alt key is down

[7] Undo flag :	0 or 1
[8] Values:	Matrix containing the values of the selected block of cells. This is the data that will replace the values in the target cells.
[9] CellSet flags:	Boolean Matrix containing the values of the CellSet property for the selected block of cells. This will replace the values of the CellSet property of the target cells.

The shift state in element 6 is intended to allow the APL programmer to implement an *insert* operation instead of a *copy* or *move* operation if required.

You may copy the contents of one block of cells to another by calling GridDropSel as a method. If so, you need only specify the *Start*, *Size* and *Target* parameters. Note that the result block becomes selected.

The Undo flag is always 1 if the event was generated by the user.

GridFCol   Property

Applies to   Grid

The GridFCol property specifies the colour of the grid lines in a Grid object

GridFCol may be a 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or it may be a scalar that defines a standard Windows colour element (see BCol for details). Its default value is 0 which obtains the colour defined for Window text.

The grid lines may be removed by setting GridFCol to the same colour as the background colour of the cells, which is defined by BCol.

GridKeyPress   Event 24

Applies to   Grid

If enabled, this event is generated when the user presses and releases a key in a Grid cell.

The GridKeyPress is reported on the Grid, *after* the KeyPress event, which is reported on the Input object associated with the current cell.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	*'GridKeyPress'* or 24
[3] Input Code:	character scalar or vector
[4] ASCII code:	integer scalar
[5] Key Number:	integer scalar
[6] Shift State:	integer scalar
[7] Input Object	character vector

For a full description of elements [3-6], see KeyPress event.

The 7th element of the event message contains the name of the Input object associated with the current cell and on which the corresponding KeyPress event has been reported.

If a callback function on the KeyPress event returns 0, the GridKeyPress event is not fired. If a callback function on the KeyPress event returns a modified KeyPress message, the GridKeyPress event is fired with the modified message and not the original one.

The default action of the GridKeyPress event is to pass its message back to the appropriate Input object to be actioned. If a callback on GridKeyPress returns 0, the keystroke will be ignored.

GridLineFCol   Property

Applies to   Grid

The GridLineFCol property specifies the colours of the grid lines in a Grid object.
GridLineFCol should be used if different coloured grid lines are required. If all the grid
lines are the same colour, use GridFCol.

GridLineFCol may be a scalar or a vector. Each item may be a 3-element vector of
integer values in the range 0-255 which refer to the red, green and blue components of
the colour respectively, or a scalar that defines a standard Windows colour element (see
BCol for details). Note that a single RGB triplet must be enclosed.

The default value of GridLineFCol is an empty numeric vector (⍬). If so, all the grid
lines are drawn using the single colour specified by GridFCol.

Elements of GridLineFCol are allocated to individual grid lines via the RowLineTypes
and ColLineTypes properties.

See also: GridLineWidth.

GridLineWidth   Property

Applies to   Grid

The GridLineWidth property specifies the widths in pixels of the grid lines in a Grid
object.

GridLineWidth may be an integer scalar or a vector. Its default value is an empty
numeric vector (⍬). If so, grid lines are drawn 1-pixel wide.

Grid lines are always displayed so that 1 pixel is drawn *within* the cell. If the width is
greater than 1 pixel, the additional pixels are drawn *between* the cells.

If an element of GridLineWidth is 0, the corresponding grid lines are not drawn.

Elements of GridLineWidth are allocated to individual grid lines via the RowLineTypes
and ColLineTypes properties.

See also: GridLineFCol.

GridLines   Property

Applies to   ListView

Requires the Windows Custom Control Library COMCTL32.DLL, Version 4.70 or higher.

The GridLines property specifies whether or not lines are displayed between items in a ListView object. GridLines applies only if the value of the View property is *'Report'*.

GridLines is a single number with the value 0 (no lines are displayed) or 1 (lines are displayed); the default is 0.

The picture below illustrates the effect on the appearance of a ListView object, of setting GridLines to 1. Note that, in this example, FullRowSelect and CheckBoxes are also set to 1.

Country	Capital	Population	Area (sq miles)
☐ Australia	Canberra	14574488	2966150
☐ Brazil	Brasilia	119070865	3286470
☐ Canada	Ottawa	22343181	3851809
☑ Denmark	Copenhagen	5119155	16614
☐ Finland	Helsinki	4787778	130120
☑ France	Paris	54334871	212973
☑ Germany	Bonn	78406000	138164
☑ Ireland	Dublin	3440427	27136
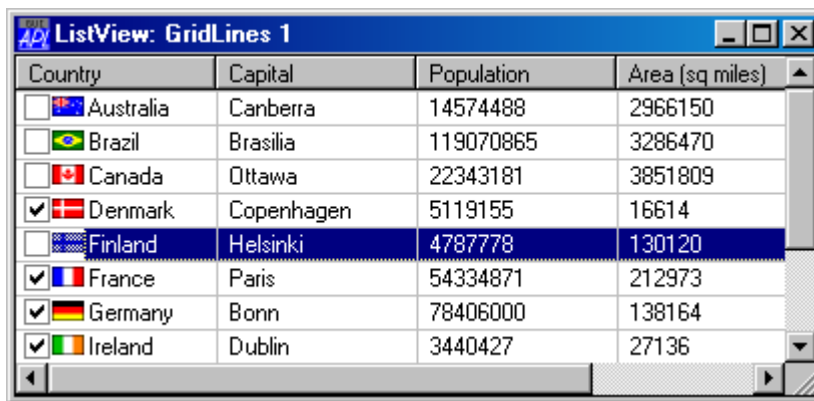
GridPaste   Event 192

Applies to   Grid

If enabled, this event is reported when the user presses Shift+Insert and there is data in the clipboard that is in a suitable format for the Grid. The default action of the event is to copy the contents of the clipboard into the currently selected block of cells, or, if no cells are selected, into the block of cells starting at the current cell (CurCell). Note that if there is a selected range of cells and the shape of the data being pasted does not exactly match the size of the selected range, the system generates a GridPasteError event in addition to the GridPaste event.

You may disable the paste facility entirely by setting the action code of the event to ¯1. You may also disable an individual paste operation by returning 0 from a callback function.

The event message reported as the result of ⎕DQ, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1] Object:		ref or character vector
[2] Event name or code:		`'GridPaste'` or 192
[3] Values:		New values (taken from the clipboard) which are to replace the existing values of the block of cells defined by Start and End.
[4] CellSet flags:		Boolean Matrix containing the new values of the CellSet property for the block of cells defined by Start and End.
[5] Start:		2-element integer vector containing the row, column address of the top left cell the selected block. If there is no selection, this is the address of the current cell(CurCell).
[6] End:		2-element integer vector containing the row, column address of the bottom right cell in the selected block. If there is no selection, this is the address of the bottom right cell of the block starting at the current cell that will be overwritten

You can replace the contents of a contiguous block of cells with the data in the clipboard, or with an arbitrary matrix of values, by calling GridPaste as a method.

If you call GridPaste with an argument of θ, the data is taken from the clipboard; otherwise the data to be pasted is specified by the *Values* and *CellSet flags* parameters.. If you omit *Start*, data is pasted into the currently selected range of cells. If there are no cells selected, data is pasted starting at the current cell (CurCell). In either case, the block of replaced cells becomes selected.

GridPasteError   Event 194

Applies to   Grid

If enabled, this event is reported when the user presses Shift+Insert and there is data in the clipboard, but the system is unable to paste the data into the Grid. This occurs if there is a currently selected block of cells whose shape does not match the shape of the data in the clipboard. It also occurs if there is no selected block of cells, and pasting the data in starting at the current cell (CurCell) would overflow the Grid. Setting the action code of this event to ¯1, or returning a 0 from a callback function attached to it, has no effect.

The event message reported as the result of ⎕*DQ*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1] Object:	ref or character vector.
[2] Event name or code:	'*GridPasteError*' or 194.
[3] Values:	Contents of the clipboard.
[4] CellSet flags:	Boolean array indicating which elements of the clipboard data are empty.
[5] Start:	2-element integer vector containing the row, column address of the top left cell in the selected block. If there is no selection, this is the address of the current cell (CurCell).
[6] End:	2-element integer vector containing the row, column address of the bottom right cell in the selected block. If there is no selection, this is the address of the bottom right cell of the block starting at the current cell that will be overwritten
[7] Error Number:	4 (*RANK ERROR*) or 5 (*LENGTH ERROR*)

GridSelect   Event 165

Applies to   Grid

If enabled, this event is reported when the user performs or cancels the selection of a
block of cells in a Grid object. This event is reported after the selection has changed.
Setting its action code to ¯1 has no effect and the result of a callback function cannot be
used to alter the selection that has been made. You may however control the user's
ability to make selections using the CellSelect property.

The event message reported as the result of $\Box DQ$, or supplied as the right argument to
your callback function, is a 4-element vector as follows :

[1] Object:		ref or character vector.
[2] Event name or code:		*'GridSelect'* or 165.
[3] Start:		2-element integer vector or matrix containing the row, column address(es) of the top left cell(s) in the selected block(s)
[4] End:		2-element integer vector or matrix containing the row, column address(es) of the bottom right cell(s) in the selected block(s)

Note that the values of Start and End are sensitive to the index origin, $\Box IO$.

If the selection is made with the mouse, the GridSelect event is reported when the left
mouse button is released. If the selection is made using the cursor keys, the GridSelect
event is reported when the Shift key is released.

The GridSelect event is also generated when the current selection is cancelled by
clicking on a cell with the mouse or by pressing a cursor key.

GripperMode   Property

Applies to   CoolBand

The GripperMode property specifies whether or not the CoolBand has a gripper bar which is used to reposition and resize the CoolBand within its parent CoolBar.

GripperMode is a character vector with the value '*Always*' (the default), '*Never*' or '*Auto*'.

If GripperMode is '*Always*' , the CoolBand displays a gripper bar even if it is the only CoolBand in the CoolBar.

If GripperMode is '*Never*' , the CoolBand does not have a gripper bar and may not be directly repositioned or resized by the user.

If GripperMode is '*Auto*' , the CoolBand displays a gripper bar only if there are other CoolBands in the same CoolBar.

Group   Object

Purpose	This object is used to group a related set of controls together visually, and to impose "radio-button" behaviour.
Parents	ActiveXControl, CoolBand, Form, Group, PropertyPage, SubForm, ToolBar, ToolControl
Children	Animation, Bitmap, Button, Calendar, Circle, ColorButton, Combo, ComboEx, Cursor, DateTimePicker, Edit, Ellipse, Font, Grid, Group, Image, ImageList, Label, List, ListView, Locator, Marker, Metafile, Poly, ProgressBar, Rect, RichEdit, Scroll, SM, Spinner, Splitter, Static, SubForm, Text, Timer, TipField, TrackBar, TreeView, UpDown
Properties	Type, Caption, Posn, Size, Coord, Border, Active, Visible, Event, Sizeable, Dragable, FontObj, FCol, BCol, Picture, CursorObj, AutoConf, YRange, XRange, Data, Attach, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, Translate, Accelerator, AcceptFiles, KeepOnClose, Redraw, MethodList, ChildList, EventList, PropList
Events	Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, FontCancel, FontOK, GotFocus, Help, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Select
Methods	Animate, ChooseFont, Detach, GetFocus, GetTextSize, ShowSIP

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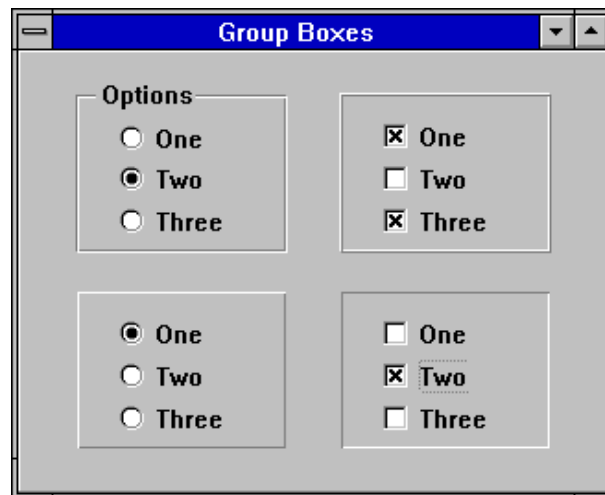
Group examples with different EdgeStyle settings

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

HAlign   Property

Applies to   Text

This property determines the horizontal alignment of text in a Text object. It is either a single integer value, or, if the Text object contains several components, a corresponding vector of such values. These may be:

0   left aligned (the left edge of the bounding box of the text is aligned on the x-co-ordinate specified by the Points property). This is the default.

1   centre aligned (the centre of the bounding box of the text is aligned on the x-co-ordinate specified by the Points property).

2   right aligned (the right edge of the bounding box of the text is aligned on the x-coordinate specified by the Points property).

Handle   Property

Applies to   ActiveXControl, Animation, Button, Calendar, ColorButton, Combo, ComboEx, CoolBar, Cursor, DateTimePicker, Edit, Font, Form, Grid, Group, Icon, ImageList, Label, List, ListView, MDIClient, Menu, MenuBar, Metafile, OLEClient, OLEServer, Printer, ProgressBar, PropertySheet, RichEdit, Scroll, SM, Spinner, Static, StatusBar, SubForm, TabBar, ToolBar, ToolControl, TrackBar, TreeView, UpDown

This is a read-only property that reports the *handle* associated with an object. For a visual object, such as a Form or a Button, this is the window handle. For a Printer, it is the *printer device context*.

This handle allows you to access the corresponding object directly with Windows API functions via ⎕NA. This facility must be used with care and the responsibility for its behaviour is entirely yours. Do NOT use it to delete an object; this will cause APL to terminate abnormally. See also NameFromHandle method.

For an example of the use of the Handle property, see function *SET_TAB_STOPS* in WTUTOR95.DWS.

HasApply   Property

Applies to   PropertySheet

The HasApply property is a Boolean value that specifies whether or not a PropertySheet has an Apply button. Its default value is 1. Note that an Apply button is only actually used if Style is *'Standard'*.

HasButtons   Property

Applies to   TreeView

The HasButtons property is a Boolean value and specifies whether or not buttons are shown in a TreeView object. If HasButtons is 1 (the default) a square button is displayed to the left of each parent item label. If the item is expanded (i.e. is children are visible) the button contains a minus sign. If the item is not expanded, (i.e. its children are hidden) the button contains a plus sign. The user can cause a parent item to expand or collapse by clicking this button.

HasCheckBox   Property

Applies to   DateTimePicker

Specifies whether or not a checkbox is displayed alongside the value in a DateTimePicker.

HasCheckBox is a single number with the value 0 (the default) or 1. If HasCheckBox is 1, the user may set or clear the checkbox to indicate whether or not the date/time displayed in the object is to apply.

If the checkbox is not set, the DateTimePicker is considered to be *empty* (the contents will be greyed out) and the value returned by the DateTime property is zilde. Note that HasCheckBox may only be set when the object is created.

HasEdit   Property

Applies to   BrowseBox

Specifies whether or not a BrowseBox has an edit field.

HasEdit is a single number with the value 0 (the default) or 1. If HasEdit is 1, the user may type in the name of a folder or other resource that is the target of the BrowseBox. If HasEdit is 0, the user must browse to it.

HasHelp   Property

Applies to   PropertyPage, PropertySheet

The HasHelp property is a Boolean value   For a PropertySheet, it determines whether or not the PropertySheet has a Help button. for a PropertyPage, HasHelp determines whether or not the Help button is active when the PropertyPage is the current page. If the HasHelp property of a PropertyPage is 0, the Help button on the parent PropertySheet will be temporarily disabled when that PropertyPage is displayed.

HasLines   Property

Applies to   TreeView

The HasLines property specifies whether or not tree lines are drawn in a TreeView object. It is a single integer with the value 0, 1 or 2:

0   No tree lines
1   Tree lines are drawn at all levels except the top level
2   Tree lines are drawn at all levels

The user can cause a parent item to expand or collapse by clicking on its corresponding tree line.

HasTicks   Property

Applies to   TrackBar

The HasTicks property specifies whether or not tick marks are drawn in a TrackBar object. It is Boolean value with a default value of 0.

The position of the tick marks in the TrackBar is determined by the TickAlign property.

HasToday   Property

Applies to   Calendar, DateTimePicker

The HasToday property specifies whether or not the Today date is displayed in the bottom left corner of a Calendar object or the drop-down calendar in a DateTimePicker..

HasToday is a single number with the value 0 (the date is *not* shown) or 1 (the date *is* shown); the default is 1.

See also CircleToday property.

Header   Property

Applies to   ListView

The Header property is Boolean and specifies whether or not a ListView object displays column titles. Its default value is 1. Header applies only if the View property is `'Report'`. The column titles are defined by the ColTitles property and their alignment by the ColTitleAlign property.

Note that Header may only be set by `⎕WC` and may not subsequently be changed.

Help   Event 400

Applies to   ActiveXControl, Animation, Button, Calendar, Circle, ColorButton, Combo, ComboEx, CoolBar, DateTimePicker, Edit, Ellipse, Form, Grid, Group, Image, Label, List, ListView, Marker, MDIClient, Poly, ProgressBar, PropertyPage, Rect, RichEdit, Scroll, SM, Spinner, Static, StatusBar, SubForm, TabBar, Text, ToolBar, ToolButton, ToolControl, TrackBar, TreeView, UpDown

If enabled, this event is reported when the user clicks on the Question (?) button in the title bar of a Form and then clicks again over the object. The presence of the Question (?) button is determined by the value of the HelpButton property. The event is also reported when the user presses function key 1 (F1).

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	'*Help*' or 400
[3] Y-coordinate:	Number
[4] X-coordinate:	Number

The y and x-coordinates refer to the position of the mouse pointer relative to the top left corner of the object and are reported in the coordinate system of that object.

HelpButton   Property

Applies to   Form, SubForm

This is a Boolean property that specifies whether or not a Question (?) button appears in the title bar of a Form or SubForm. However, this does not apply if the Form has a maximise or minimise button which both take precedence. The user may obtain help by clicking on the Question (?) button and then on a control in the Form. It is up to you to provide the help by responding to the Help event on the control. The default value of HelpButton is 0.

HelpFile   Property

Applies to   ActiveXControl, OCXClass, OLEClient

This property is a character vector that specifies the pathname of a Windows help file associated with a particular object.

For an OCXClass or OLEClient object, the HelpFile property is read-only

HideComment   Event 224

Applies to   Grid

If enabled, a HideComment event is generated just before a comment window is hidden as a result of the user moving the mouse-pointer away from a commented cell.

The event message reported as the result of $\Box DQ$, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	*'HideComment'* or 224
[3] Row:	integer
[4] Column:	integer

You may prevent the comment from being hidden by returning 0 as the result of a callback function.

Invoked as a method, HideComment is used to hide a comment that has previously been displayed by ShowComment. For example, the following expression hides the comment associated with the cell at row 2, column 1.

```
F.G.HideComment 2 1
```

If HideComment is called with an argument of 0, all comments are hidden.

Hint   Property

Applies to   Animation, Button, Calendar, ColorButton, Combo, ComboEx,
                        DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView,
                        MDIClient, MenuItem, ProgressBar, PropertyPage, PropertySheet,
                        RichEdit, Scroll, SM, Spinner, Static, StatusBar, SubForm, TabBar,
                        ToolBar, ToolButton, TrackBar, TreeView, UpDown

The Hint property is a character vector that specifies a *help* message that is to be
displayed when the user positions the mouse pointer over the object. The Hint is
displayed in the object specified by its HintObj property. A StatusField is often used for
this purpose.

HintObj   Property

Applies to   Animation, Button, Calendar, ColorButton, Combo, ComboEx,
                        DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView,
                        MDIClient, MenuItem, ProgressBar, PropertyPage, PropertySheet,
                        RichEdit, Root, Scroll, SM, Spinner, Static, StatusBar, SubForm,
                        TabBar, ToolBar, ToolButton, TrackBar, TreeView, UpDown

The HintObj property is a character vector that specifies the name of an object in which
the "help" message defined by the Hint property is to be displayed. This message is
displayed when the user positions the mouse pointer over the object. The Hint is
displayed by automatically setting the Caption or Text property of the object named by
HintObj. The following types of object can therefore be used to display Hints: Button,
Edit, Label, Combo, Group, Form, Label, Menu, MenuItem, StatusField, SubForm and
Text. For a StatusField that has both Caption and Text properties, the text property is
used for displaying hints.

When the user moves the mouse pointer away from the object, the Caption or Text
property of the object named by HintObj is reset to an empty vector. Note that if
HintObj is empty, its value is **inherited** from its parent. Thus setting HintObj on a Form
defines the default location for displaying Hints for all the controls in that Form. Setting
HintObj on Root defines the default location for hints for the entire application.

HotSpot   Property

Applies to   Cursor

This property specifies the point within a Cursor object that registers the cursor's position over another object. The mouse position, which is reported by various events, is actually the position of the cursor's HotSpot over the object in question.

HotSpot is a 2-element numeric vector that specifies the y-position and x-position of the hotspot within the cursor. A value of (0 0) specifies the top-left corner of the cursor; (31 31) specifies the bottom right corner of the cursor. The default value of HotSpot is (15 15).

HotTrack   Property

Applies to   TabControl

The HotTrack property specifies whether or not the tabs or buttons in a TabControl object (which are represented by TabButton objects), are automatically highlighted by the mouse pointer.

HotTrack is a single number with the value 0 (no highlighting) or 1. The default is 0.

If HotTrack is 1 and the Style property of the TabControl is *'Tabs'* or *'Buttons'*, the text defined by the Caption property of the TabButton is highlighted when the mouse pointer is placed over the tab or button. If Style is *'FlatButtons'*, the button is highlighted by being *raised*.

The value of HotTrack is effective only when the object is created with ⎕*WC*.

HScroll   Property

Applies to   Combo, ComboEx, Edit, Form, Grid, ListView, RichEdit, Scroll, StatusBar, SubForm, TabBar, ToolBar, TrackBar, UpDown

For most objects to which it applies, HScroll specifies whether or not a horizontal scrollbar is provided.

When applied to a Combo, or to an Edit object with Style '*Single*' (i.e. a single-line edit field), the value 0 inhibits scrolling, and prevents the user from entering more data when the field is full. If instead it has the value ¯2, the field is scrollable, and the length of data that may be entered is not limited by the length of the field.

When applied to an Edit object with Style '*Multi*' (i.e. a multi-line text box), the value 0 inhibits scrolling, and causes individual lines to be "word-wrapped". The values ¯2 and ¯1 enable sideways scrolling, and permit individual lines to exceed the width of the object. The value ¯1 means that a horizontal scrollbar is provided.

For a Scroll object, the scrollbar is horizontal if HScroll is ¯1 and vertical if HScroll is 0. For a Form, a horizontal scrollbar is provided if HScroll is set to ¯1. The default value is 0 (no scrollbar).

For a StatusBar, TabBar or ToolBar with Align set to Top or Bottom, HScroll determines whether or not a horizontal scrollbar is provided and how the object positions its children. If HScroll is 0 (the default) the object organises its children in multiple rows and does not provide a scrollbar. If HScroll is ¯1 or ¯2, the object organises its children in a single row and provides a mini scrollbar to allow those positioned beyond the right edge of the object to be scrolled into view. If HScroll is ¯1, the scrollbar is always shown. If HScroll is ¯2, it is only shown when needed.

For a Grid, HScroll may be 0 (no horizontal scrollbar), ¯1 (scrollbar is displayed when required), ¯2 (same as ¯1) or ¯3 (scrollbar is always displayed).

HScroll   Event 39

Applies to   Form, SubForm

If enabled, this event is generated when the user attempts to move the thumb in a horizontal scrollbar in a Form or SubForm. This event occurs only in a Form whose HScroll property is set to ¯1 and is distinct from the Scroll event that is generated by a Scroll object. The event may be generated in one of three ways:

a)   dragging the thumb.

b)   clicking in one of the "arrow" buttons situated at the ends of the scrollbar. This is termed a small change, the size of which is defined by Step[3].

c)   clicking in the body of the scrollbar. This is termed a large change, the size of which is defined by Step[4].

The event message reported as the result of $\Box DQ$, or supplied as the right argument to your callback function, is a 4-element vector as follows:

[1] Object: ref or character vector

[2] Event code: *'HScroll'* or 39

[3] Scroll Type: numeric

[4] Position: numeric

The value of Scroll Type is 0 (drag), 1 or ¯1 (small change) or 2 or ¯2 (large change). The sign indicates the direction.

The value of Position is the new (requested) position of the thumb. Notice however, that the event is generated **before** the thumb is actually moved. If your callback function returns a scalar 0, the position of the thumb will remain unaltered.

Icon Object

Purpose	This object defines an icon.
Parents	ActiveXControl, CoolBand, Form, Grid, ImageList, ListView, OLEServer, Printer, ProgressBar, PropertyPage, PropertySheet, RichEdit, Root, StatusBar, SubForm, SysTrayItem, TCPSocket, ToolBar, ToolControl, TrackBar, TreeView, UpDown
Children	Timer
Properties	Type, File, Bits, CMap, Mask, Style, KeepBits, Event, Data, Handle, Accelerator, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, Select
Methods	Detach, FileRead, FileWrite

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

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

If the value of the File property is set by $\Box WS$, no immediate action is taken, but the corresponding file may subsequently be read or written using the FileRead or FileWrite methods.

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

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

If an element of Mask is 0, the corresponding element of Bits defines the colour of the resulting pixel that is displayed on the screen. If an element of Mask is 1, the resulting pixel that is displayed on the screen is either the current background colour or is a new colour chosen by Windows to be visible against the background. A non rectangular icon is obtained by setting those elements of Bits and Mask that you want to exclude from the shape to be 0 and 1 respectively. Under Windows 3.1, the size of Bits and Mask **must** be 32 x 32.

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

IconObj   Property

Applies to   Form, MDIClient, Root, SubForm, SysTrayItem, TabBar, ToolBar

This property is used to specify a *large* and *small* icon for a Form or SubForm, or for the Root object which represents your application as a whole. Its value is either a character scalar or vector containing the name of an Icon object, or a 2-element vector of character vectors containing the names of 2 Icon objects.

If empty (the default value), the standard "Dyalog APL" icon is used.

The large and small icons are supplied to the Operating System which uses them as and when is appropriate. Normally, the large icon is of size 32x32 and the small icon is 16x16. If you specify an icon of a different size, the Operating System will scale it as appropriate.

For an MDIClient, the IconObj property has no direct use, but is **inherited** by all its child SubForms. Thus if you want all your child SubForms to use the same icons, you need only define them once for the MDIClient.

Idle   Event 130

Applies to   Root

If enabled, this event is generated whenever APL looks to see if there is an event on the queue and finds it empty. Its purpose is to allow an application to perform some background processing when the user is not doing anything. It is unwise to use this event directly from the Session as it will occur repeatedly and may lock you out.

The event message reported as the result of □*DQ*, or supplied as the right argument to your callback function is a 2-element vector as follows:

[1] Object: ref or character vector

[2] Event code: '*Idle*' or 130

IDNToDate   Method 263

Applies to   Calendar, DateTimePicker, Root

This method is used to convert a date from an IDN into □*TS* format (year, month, day). The corresponding day of the week is also obtained.

The argument to IDNToDate is a single item as follows:

[1] IDN: Integer

The result is a 4-element integer vector containing the year, month, day, and weekday corresponding to the IDN that was specified.

The value of the 4th element, weekday, is an integer in the range 0-6 that specifies on which day of the week the specified date falls (0=Monday).

Example:

```
    F.C.IDNToDate 36048
1998 9 11 4
```

Image   Object

Purpose	Positions bitmaps and icons within an object.
Parents	ActiveXControl, Bitmap, Form, Grid, Group, Metafile, Printer, PropertyPage, Static, StatusBar, SubForm, ToolBar, ToolControl
Children	Timer
Properties	Type, Points, Coord, Visible, Event, Dragable, Picture, OnTop, AutoConf, Data, EdgeStyle, Size, Accelerator, AcceptFiles, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, DragDrop, Help, MouseDblClick, MouseDown, MouseMove, MouseUp, Select
Methods	Detach

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

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

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

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

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

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

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

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

Examples

First make a Form, then make two Bitmaps:

```
'F' ⎕WC 'Form'
'YES' ⎕WC 'Bitmap' 'C:\WDYALOG\WS\YES'
'NO'  ⎕WC 'Bitmap' 'C:\WDYALOG\WS\NO'
```

Display the `YES` Bitmap at (20,10)

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

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

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

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

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

ImageCount   Property

Applies to   ImageList

The ImageCount property is a read-only property that reports the number of images in an ImageList object. It is an integer scalar.

ImageIndex   Property

Applies to   ComboEx, CoolBand, ListView, Menu, MenuItem, TabButton, ToolButton, TreeView

For a ComboEx, ListView or TreeView, the ImageIndex property maps bitmapped images in an ImageList to items. ImageIndex is an integer vector whose length is the same as the number of items in the object. See also SelImageIndex

For a CoolBand, MenuItem, TabButton or ToolButton, ImageIndex specifies the picture to be displayed in the object. In these cases, ImageIndex is a single integer value.

ImageIndex is $\Box IO$ dependent.

ImageList   Object

Purpose	The ImageList object represents a set of bitmapped images.
Parents	ActiveXControl, CoolBand, CoolBar, Form, Group, ListView, OLEServer, PropertyPage, Root, SubForm, TabControl, TCPSocket, ToolBar, ToolControl, TreeView
Children	Bitmap, Cursor, Icon, Timer
Properties	Type, Size, Event, Data, Handle, Translate, ImageCount, Masked, MapCols, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create
Methods	Detach

An ImageList object represents an array of bitmapped images which are used to depict items in a ComboEx, ListView, or TreeView object, or the images for a CoolBand, MenuItem, TabControl or ToolControl.

Making an ImageList is a 2-step process. First, you create an (empty) ImageList specifying its Size and Masked properties. The former establishes the size of each of the bitmapped images in the array. The Masked property specifies whether the ImageList is to contain masked images (Icons and Cursors) or unmasked images (Bitmaps). The default is 1 (Icons). Note that these properties must be established when the ImageList is created by `⎕WC` and may not subsequently be changed using `⎕WS`.

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

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

Note that you can associate a set of bitmap images contained in a DLL by specifying the name of the DLL and the resource number of the bitmap. For example:

```
'F.IL'⎕WC'ImageList'('Masked' 0)('MapCols' 1)
'F.IL.'⎕WC'Bitmap'('ComCtl32' 120)⍺ STD_SMALL

'F.TB.IL'⎕WC'ImageList'('Masked' 0)('Size' 24 24)
'F.TB.IL.'⎕WC'Bitmap'('ComCtl32' 121)⍺ STD_LARGE
```

Notice that when making Bitmaps or Icons as children of an ImageList, it is normally not necessary to *name* them because they are subsequently referenced only via the ImageIndex and SelImageIndex properties and not by name. The number of images in an ImageList is given by the read-only property, ImageCount.

An ImageList is *associated* with another object via its ImageListObj property.

ImageListObj   Property

Applies to   ComboEx, CoolBar, ListView, Menu, TabControl, ToolControl, TreeView

The ImageListObj property is a simple character vector or a vector of character vectors that specifies the name(s) of ImageList objects that are associated with an object.

For a ComboEx or TreeView object, the ImageListObj property specifies the name of a single ImageList object that contains a set of images to be displayed alongside its Items. The image(s) displayed by a particular item in its normal (unselected) and selected states are specified by the corresponding element of the ImageIndex and SelImageIndex properties respectively.

For a CoolBar, Menu, and TabControl objects, the ImageListObj property specifies the name of a single ImageList object that contains a set of images for its CoolBand, MenuItem and TabButton children respectively. For a ToolControl, ImageListObj may specify up to three ImageList objects that correspond to each of the three different states, normal, highlighted (hot) and inactive, of its ToolButton children. In all these cases, individual images are mapped to the child objects by their ImageIndex property.

For a ListView object, either one or two ImageList objects may be specified. The first ImageList contains the *large icon* set of images. the second contains the *small icon* set. The set that is used is determined by the value of the View property. The mapping between the set of images in the ImageList and items in the object is determined by the ImageIndex property.

Indents   Property

Applies to   ComboEx

Specifies the amount by which items in a ComboEx object are indented.

Indents may be an integer scalar or a vector with the same number of elements as there are items in the ComboEx. Its default value is 0.

The unit of indenting is 10 pixels. For example, if there are 3 items and Indents is (0 1 2), the items will be indented by 0, 10 and 20 pixels respectively.

Index   Property

Applies to   Combo, ComboEx, CoolBand, FileBox, Grid, List, TreeView

For a List and a Combo with Style '*Simple*', this property specifies the position of the data in the list box as a positive integer value. If Index has the value "n", it means that the "nth" item in Items is displayed on the top line in the list box. Note that Index for a Combo or List cannot be set using □*WC*. The value of Index in a Combo with a drop-down list box (Style '*Drop*' or '*DropEdit*') is always equal to □*IO*.

For a Grid, Index is a 2-element vector that specifies the row and column number of the cell that is currently in the top left corner of the Grid.

For a TreeView, Index is a positive integer value that specifies which item appears at the top of the TreeView window.

For a FileBox, the Index property determines which of the Filters is initially selected.

For a CoolBand, the Index property specifies the position of the CoolBand within its parent CoolBar, relative to the other CoolBands in the CoolBar.

The value of Index is dependent on □*IO*, and its default value is equal to □*IO*.

IndexChanged   Event 210

Applies to   Grid

If enabled, this event is reported when the value of the Index property of a Grid has changed as a result of user interaction. The event is reported *after* the Grid has been scrolled. You may not modify or nullify the operation with a 0-return callback and you may not call IndexChanged as a method or generate this event using *⎕NQ*. To cause a Grid to scroll, use *⎕WS* to set its Index property.

The event message reported as the result of *⎕DQ*, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1] Object:		ref or character vector
[2] Event name or code:		*'IndexChanged'* or 210
[3] Row:		Integer.
[4] Column:		Integer.

The Row and Column reported by the event refer to the new value of the Index property.

Input   Property

Applies to   Grid

This property specifies the name or names of objects to be associated with cells in a Grid. These objects may be of type Button, ColorButton, Combo, Edit, Label, Spinner or TrackBar. In addition, the Input property my specify instances of OCXClass objects (ActiveX controls) and NetClient objects (.NET classes).

The Input property is either a simple character scalar or vector, or a vector of character vectors. If it is simple, it specifies the name of a single object to be associated with all of the cells in the Grid. If it is nested, it specifies a set of objects that are mapped to individual cells through the CellTypes property.

When a cell becomes the current cell, its value (defined by the appropriate element of the Values property) defines the value of a *corresponding property* of the associated object The property that corresponds to the value in the cell, depends upon the Type of the associated object as shown in the following table:

Associate Object Type	Corresponding Property
Label	Text
Edit	Value
Combo	Text
Button (Style Push)	Caption
Button (Style Radio or Check)	State
ColorButton	CurrentColor
Spinner	Value
TrackBar	Thumb
OCXClass	(specified by InputProperties)
NetClient	(specified by InputProperties)

In effect, the user inputs a new value into the current cell by changing the corresponding property of its associated object. An associated object may be a *fixed* object that is *external* to the Grid or a *floating* object that moves automatically from cell to cell. The latter is achieved by creating the associated object as a *child* of the Grid.

If the associated object is an Edit or Combo, the user may change the Text property of the object by typing, or, in the case of a Combo, by selecting an item from a list. The new value of the Text property is then used to update the value in the cell (defined by the Values property of the Grid) when the user moves on. If the associated object is a radio button, the value in the cell (0 or 1) is reflected by the State property of the Button. The user may click the Button on and off, changing its State and thereby the corresponding value in the cell.

If the associated object is a ColorButton, the corresponding elements of the Values property contain 3-element integer vectors which specify the RGB colour values.

If the associated object is an instance of an OCXClass object (ActiveX control) or a NetClient object (.NET class), the Grid uses the *default* property of the external object if it has one. Alternatively, the InputProperties property is used to specify which property (or properties) of the external object are to be mapped to elements of Values. If more than one property is specified, elements of Values are vectors.

If there is no object associated with a cell, or if its associated object is a Label or a Button with Style Push, the cell is protected and may not be changed by the user. When the current cell is thus protected, the corresponding row and column titles are not indented as they are when the cell may be edited.

If the associated object is a numeric Edit or Label (FieldType Numeric, LongNumeric, Currency, Date, LongDate or Time) the contents of the cell are formatted accordingly, even when it is not the current cell. Thus a cell associated with a Label with FieldType Date, always displays as a date.

If the associated object is a Combo or Button, the appearance of a non-current cell is defined by the corresponding element of the ShowInput property.

The following example illustrates the use of different types of object specified by the Input and ShowInput properties.

Surname	Job Title		Region		Salary	Permanent
Brown	Manager	±	South	±	$64000.00	☒
Jones	Project Leader	±	South	±	$43250.00	☒
Green	Consultant	±	South	±	$45000.00	☐
Black	Programmer	±	East	±	$30000.00	☒
White	Assistant	±	Central	±	$40000.00	☐

Different Input Objects

InputMode   Property

Applies to   Grid

This property determines editing behaviour and the action of the cursor movement keys when the user changes the contents of a Grid using a *floating* Edit or DropEdit Combo control.

InputMode is a character vector with one of the following values:

`'Scroll'`	The cursor keys move around the Grid; the user may switch to *InCell* mode.
`'InCell'`	The cursor keys move within the Input object; the mode reverts to *Scroll* when the user selects a new cell.
`'AlwaysScroll'`	The cursor keys move around the Grid; the user may not switch to *InCell* mode.
`'AlwaysInCell'`	The cursor keys move within the Input object, even when the user moves to a new cell
`'AutoEdit'`	See below

By default, the input mode is *Scroll*. In this mode, cursor movement keys are actioned by the Grid itself and used to move from cell to cell. The user may switch to *InCell* mode by double-clicking or by pressing the key defined by InputModeKey (the default is "F2").

In *InCell* mode, all cursor movement keys are actioned by the Input object and typically move the cursor around *within* the Input object and do not switch between cells. When the user switches to a different cell, InputMode reverts to *Scroll* mode

If InputMode is *AlwaysScroll* or *AlwaysInCell*, the user remains permanently in either *Scroll* or *InCell* mode respectively.

If InputMode is `'AutoEdit'`, the behaviour of a cell that contains a floating Edit field is as follows:

When the user enters the cell, the contents are selected (and highlighted).At this stage, the cursor movement keys move to an adjacent cell
If the user presses a (valid) data key, that character replaces the current contents of the cell.

If the user presses F2 (or the key defined by the InputModeKey property), the data is de-selected and unhighlighted and the cursor is placed at the rightmost end of the data.

In either case, the left and right cursor keys now move the cursor within the current data string, but skip to the adjacent cell from the beginning or end of the data. This behaviour differs from *InCell* mode in which the cursor movement keys *stick* at the end of the data.

InputModeKey   Property

Applies to   Grid

This property defines the keystroke used to switch from Scroll mode to InCell mode in a Grid. It applies only where the Grid has a *floating* Edit control. See the description of the InputMode property for further details.

The InputModeKey property is specified (in the same way as the Accelerator property) as a 2-element vector of integer values containing the key number and shift state respectively. Its default is `(113 0)` which is "F2".

As an example, if you wanted to use Ctrl+Shift+a to switch modes, you would set InputModeKey to `(65 3)`. 65 is the key number for "a" and 3 means Shift (1) + Ctrl (2).

InputProperties   Property

Applies to   Grid

The InputProperties property is a vector of character vectors that specifies the names of properties of an OCXClass (ActiveX Control) or .NET Class that are to be mapped to the Values property in a Grid.

When an ActiveX Control or .NET Class is used as a child of the Grid, InputProperties is used to specify how the value in each Grid cell corresponds to the value of one or more properties of the child object.

For example, suppose there is a third-party ActiveX Control that displays a playing card. The Control has two properties named Suit and Value that specify the suit (1=clubs, 2=diamonds, 3=hearts, 4=spades) and card value (1="Ace", 2="2", …11="Jack",…) respectively. To display these cards in a Grid, the InputProperties property may be set to (*'Suit' 'Value'*) and each element of the Values property must be a 2-element integer vector specifying the suit and value of the corresponding card.

```
'CARDS'⎕WC'OCXClass' '...'
'F'⎕WC'Form'
'F.G'⎕WC'Grid'
'F.G.card'⎕WC'CARDS'
F.G.Input←'F.G.card'
F.G.InputProperties←'Suit' 'Value'
F.G.Values←⍳4 13
```

If InputProperties is not specified, the *default* property of the ActiveX Control or .NET Class is used.

InstanceMode   Property

Applies to   OLEClient

The InstanceMode property specifies how APL attempts to connect the OLEClient to an OLE Server.

InstanceMode is a character vector that may be `'ExistingFirst'` (the default), `'ExistingOnly'` or `'New'`. Its value is effective only when the object is created with `⎕WC`. Changing InstanceMode with `⎕WS` has no effect.

If InstanceMode is `'ExistingFirst'`, APL attempts first to connect to a running instance of the OLE Server. If there is no running instance, it starts the OLE server to obtain a new object.

If InstanceMode is `'ExistingOnly'`, APL attempts to connect to a running instance of the object. If there is no running instance, it fails with a `DOMAIN ERROR`.

Note that in either case, if there is more than one instance running, there is no way to predict to which instance APL will be connected.

If InstanceMode is `'New'`, APL attempts to start the OLE Server to obtain a new object, whether or not the OLE Server is already running. However, if the OLE Server has registered itself as a single instance object and is already running, APL will be connected to it, and a second instance of the Server will not in fact be started.

Interval   Property

Applies to   Timer

The Interval property specifies the frequency with which a Timer object generates Timer events. It is an integer value specified in milliseconds and has a default of 1000.

Setting Interval to 0 effectively de-activates the Timer.

Italic   Property

Applies to   Font

This property specifies whether or not a font represented by a Font object is italicised or not. It is either 0 (normal) or 1 (italic). There is no default; the value of this property reflects the characteristic of the selected font.

ItemDblClick   Event 342

Applies to   ListView, TreeView

If enabled, this event is reported when the user double-clicks a mouse button when the mouse pointer is over an item in a ListView or TreeView object. This event is reported for information only and may not be controlled in any way using a callback function. Generating the event with *⎕NQ*, or calling it as a method, has no effect.

The event message reported as the result of *⎕DQ*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	*'ItemDblClick'* or 342
[3] Item number:	Integer. The index of the item.
[4] Mouse button:	Integer.
[5] Shift state:	Integer. Sum of 1=Shift key, 2=Ctrl key, 4=Alt key
[6] Position:	Integer. Indicates the position of the mouse-pointer within the item. It is either 2 (over the icon), 4 (over the label), 8 (over the line) or 16 (over the symbol).

ItemDown Event 340

Applies to ListView, TreeView

If enabled, this event is reported when the user depresses a mouse button when the mouse pointer is over an item in a ListView or TreeView object. This event is reported for information only and may not be controlled in any way using a callback function. Generating the event with `⎕NQ`, or calling it as a method, has no effect.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	`'ItemDown'` or 340
[3] Item number:	Integer. The index of the item.
[4] Mouse button:	Integer.
[5] Shift state:	Integer. Sum of 1=Shift key, 2=Ctrl key, 4=Alt key
[6] Position:	Integer. Indicates the position of the mouse-pointer within the item. It is either 2 (over the icon), 4 (over the label), 8 (over the line) or 16 (over the symbol)

Items Property

Applies to Combo, ComboEx, List, ListView, Spinner, TreeView

This property specifies the list of items for a Combo, ComboEx or List object from which the user may choose. Each item is represented by a row in the listbox.

The value of Items is a text array. It is normally specified as a vector of character vectors each of which represents an item. For a Combo, List or Spinner, Items may also be a matrix whose rows specify items. If a character scalar or simple vector is specified, it is treated as a single item.

An empty character vector (which is the default) is treated the same as a vector of blanks, and represents one item. A zero-length vector of vectors or an empty matrix represents 0 items. The Items property returns an array of the same structure as was assigned by `⎕WC` or `⎕WS`.

ItemUp   Event 341

Applies to   ListView, TreeView

If enabled, this event is reported when the user releases a mouse button when the mouse pointer is over an item in a ListView or TreeView object. This event is reported for information only and may not be controlled in any way using a callback function. Generating the event with `⎕NQ`, or calling it as a method, has no effect.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	`'ItemUp'` or 341
[3] Item number:	Integer. The index of the item.
[4] Mouse button:	Integer.
[5] Shift state:	Integer. Sum of 1=shift key, 2=Ctrl key, 4=Alt key
[6] Position:	Integer. Indicates the position of the mouse-pointer within the item. It is either 2 (over the icon), 4 (over the label), 8 (over the line) or 16 (over the symbol)

Justify   Property

Applies to   Button, Edit, Label, Spinner, TabControl

For a Button, Edit, Label and Spinner, this property determines the manner in which text is justified within the object. It is a character vector that may take the value `'Left'` (the default), `'Centre'` or `'Right'`. The keyword `'Centre'` may also be spelled `'Center'`.

When applied to an Edit object with Style `'Multi'`, a value of `'Centre'` or `'Right'` forces word-wrapping and disables horizontal scrolling. Note that Justify only applies to a multi-line edit field. If you specify a value for Justify in a 1-line edit field (Style `'Single'`), it will be ignored.

For a TabControl, Justify may be `'Right'` (which is the default) or `'None'` or empty. If Justify is `'Right'`, the TabControl increases the width of each tab, if necessary, so that each row of tabs fills the entire width of the tab control. Otherwise, if Justify is empty or 'None', the rows are ragged.

With the exception of Label and TabControl objects, Justify may only be specified when the object is created using `⎕WC`.

KeepBits   Property

Applies to   Bitmap, Cursor, Icon

This property is be used to control the way that a Bitmap, Cursor and Icon objects are stored in the workspace.

When you create a Bitmap, Icon or Cursor using `⎕WC`, APL asks Windows to allocate a corresponding bitmap, icon or cursor *resource*. This resource is allocated in Windows memory. If APL were to hold the values of the *image properties* (CBits, Bits and CMap for a Bitmap; Bits, CMap and Mask for Cursor and Icon objects) internally in the workspace, this data would be duplicated. For large bitmaps this would have a serious impact on memory utilisation and may affect performance. The KeepBits property is provided to allow you to control whether or not APL retains the values of the *image properties* in the workspace, so that you can choose a strategy to suit your configuration and requirements. KeepBits may take the value 0 or 1.

If KeepBits is 0 the values of the *image properties* are **not** stored internally in your workspace. If you save a workspace containing a Bitmap, Cursor or Icon object, the corresponding Windows resource is automatically re-allocated when the workspace is loaded by referring to the associated file. This is the file whose full pathname is defined by the value of the object's File property. It follows that if you adopt this strategy, you must ensure that the File property is set correctly. If APL cannot find the file when the workspace is `)LOAD`ed, it cannot re-create the object, and you will get a *VALUE ERROR* when you subsequently refer to it. A further consideration is the effect on `⎕WG`. If KeepBits is 0, and you execute `⎕WG 'CBits'` or `'Bits'` or `'CMap'` or `'Mask'`, APL obtains these values by requesting the data from Windows.

If KeepBits is set to 1, the contents of the *image properties* are stored in the workspace, thus duplicating the information which is held by Windows itself. If you save a workspace containing a Bitmap, Cursor or Icon the corresponding Windows resource is automatically re-allocated from the *image properties* when the workspace is loaded. The value of the File property is ignored. When you execute `⎕WG 'CBits'` or `'Bits'` or `'CMap'` or `'Mask'`, APL generates the result directly from the stored values held (internally) in the workspace.

KeepOnClose   Property

Applies to   ActiveXContainer, ActiveXControl, Animation, Bitmap, BrowseBox,
Button, Calendar, Circle, Clipboard, ColorButton, Combo, ComboEx,
CoolBand, CoolBar, Cursor, DateTimePicker, Edit, Ellipse, FileBox,
Font, Form, Grid, Group, Icon, Image, ImageList, Label, List,
ListView, Locator, Marker, MDIClient, Menu, MenuBar, MenuItem,
Metafile, MsgBox, OCXClass, OLEClient, OLEServer, Poly, Printer,
ProgressBar, PropertyPage, PropertySheet, Rect, RichEdit, Root,
Scroll, Separator, SM, Spinner, Splitter, Static, StatusBar, StatusField,
SubForm, SysTrayItem, TabBar, TabBtn, TabButton, TabControl,
TCPSocket, Text, Timer, TipField, ToolBar, ToolButton,
ToolControl, TrackBar, TreeView, UpDown

This property is either 0 or 1 and determines how the object is treated when its parent
Form (or, in the case of a Form, the Form itself) is closed by the user, receives a Close
event from □*NQ*, or when Close is called as a method.

If KeepOnClose is 1 (for the object itself **and** for all its parents) when its parent Form is
closed, the object changes from being a GUI object to a pure namespace. For example,
the Type of a Button will change from '*Button*' to '*Namespace*'. Effectively, the
GUI component of the object is discarded but its Namespace component (and any
variables, functions, operators and other namespaces that it contains) remains intact.
Monadic □*WC* may subsequently be used to re-attach the GUI component to the object.

Note that the default value of KeepOnClose depends upon the way in which a GUI
object was created with □*WC*. If a GUI object is created by dyadic □*WC*, KeepOnClose
defaults to 0. If a GUI object is attached by monadic □*WC*, its KeepOnClose property
defaults to 1.

KeyError   Event 23

Applies to   Edit, Spinner

If enabled, this event is generated when the user presses a key that is invalid according to the FieldType of the object. This event is reported for information only. You may not disable or modify it using the result of a callback function. If the KeyPress event is enabled too, KeyPress is reported before KeyError.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	`'KeyError'` or 23
[3] Input Code:	character scalar or vector
[4] ASCII code:	integer scalar
[5] Key Number:	integer scalar
[6] Shift State:	integer scalar

The input code contains the $\square AV$ character or command to which the keystroke maps through the Input Translate Table.

The ASCII code is a number in the range 0-255 which specifies the ASCII character that would normally be generated by the keystroke, and is independent of the Input Translate Table. If there is no corresponding ASCII character, the ASCII code reported is 0.

The key number is the physical key number reported by Windows when the key is pressed.

The Shift State indicates which (if any) of the Shift, Ctrl and Alt keys are down at the same time as the key is pressed. It is the sum of the following numbers :

Shift key down	:	1
Ctrl key down	:	2
Alt key down	:	4

Thus a Shift State of 3 indicates that the user has pressed the key in conjunction with both the Shift and Ctrl keys. A Shift State of 0 indicates that the user pressed the key on its own.

KeyPress   Event 22

Applies to   ActiveXControl, Animation, Button, Calendar, ColorButton, Combo, ComboEx, DateTimePicker, Edit, Form, Grid, Group, List, ListView, MDIClient, ProgressBar, PropertyPage, RichEdit, Scroll, Spinner, SubForm, TrackBar, TreeView

If enabled, this event is generated when the user presses and releases a key on the keyboard. It is reported for whichever object has the keyboard focus at the time.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	*'KeyPress'* or 22
[3] Input Code:	character scalar or vector
[4] ASCII code:	integer scalar
[5] Key Number:	integer scalar
[6] Shift State:	integer scalar

The input code contains the $\square AV$ character or command to which the keystroke maps through the Input Translate Table.

The ASCII code is a number in the range 0-255 which specifies the ASCII character that would normally be generated by the keystroke, and is independent of the Input Translate Table. If there is no corresponding ASCII character, the ASCII code reported is 0.

The key number is the physical key number reported by Windows when the key is pressed.

The Shift State indicates which (if any) of the Shift, Ctrl and Alt keys are down at the same time as the key is pressed. It is the sum of the following numbers :

Shift key down	:	1
Ctrl key down	:	2
Alt key down	:	4

Thus a Shift State of 3 indicates that the user has pressed the key in conjunction with both the Shift and Ctrl keys. A Shift State of 0 indicates that the user pressed the key on its own.

For example, pressing keys in Form *'Form1'* would generate the following event messages :

```
Form1  22 a 97 65 0       ⍝ user pressed "a"

Form1  22 A 65 65 1       ⍝ user pressed "Shift-a"

Form1  22  UC  0 38 0     ⍝ user pressed "Up Cursor"
```

Label   Object

Purpose   Displays static text.

Parents   ActiveXControl, CoolBand, Form, Grid, Group, PropertyPage, SubForm, ToolBar, ToolControl

Children   Circle, Cursor, Ellipse, Font, Marker, Poly, Rect, Text, Timer

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

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

Methods   Animate, ChooseFont, Detach, GetFocus, GetTextSize, ShowSIP

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

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

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

By default, the value of the EdgeStyle property for a Label is *'None'* and the value of BCol is 0 which implies Button Face colour. You can change its appearance by setting EdgeStyle and/or BCol to different values.

The following illustration shows a "default" Label, a Label with a border, and a Label that uses a Courier New font. The examples are drawn on Form which has a EdgeStyle property of *'Default'*. The code to produce this example is as follows :

```
'T'⎕WC'Form' 'Labels'(40 10)(30 50)
 L←'Chateau Baleau'
 'T.L1'⎕WC'Label'L(15 8)
 'T.L2'⎕WC'Label'L(15 62)('Border' 1)
 'T.F' ⎕WC'Font' 'Courier New' 32 1 0 0 700
 'T.L3'⎕WC'Label'L(60 8)('FontObj' 'T.F')
```
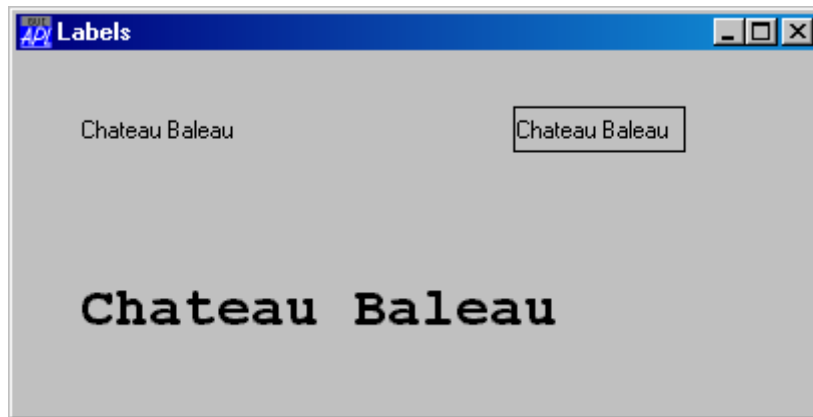

The second illustration shows some 3-Dimensional Label objects drawn on a default Form. The APL code to produce this example is as follows.

```
L←'Chateau Baleau'
'T'⎕WC'Form' '3-Dimensional Labels' (40 10)(30 50)
'T.L1'⎕WC'Label'L(15 8)('EdgeStyle' 'Plinth')
'T.L2'⎕WC'Label'L(15 62)('Border' 1)
                ('EdgeStyle' 'Recess')
'T.F' ⎕WC'Font' 'Courier New' 37 0 0 0 700
'T.L3'⎕WC'Label'L(60 8)('FontObj' 'T.F')
        ('EdgeStyle' 'Groove')('Justify' 'Centre')
```

LastError   Property

Applies to   ActiveXControl, OLEClient, OLEServer, Root

The LastError property provides information about the most recent error reported by
OLE. You may use this property to report an error from an OLEServer or
ActiveXControl to a host application.

LicenseKey   Property

Applies to   OCXClass

The LicenseKey property is a character string that contains the license key for an
ActiveX control.

If an ActiveX control requires a license key, it must be specified by an application when
it creates an instance of the control. Typically, the license key is *required* only by the
run-time version of an ActiveX control, and is *made available* to an application by the
development version of the control.

Limits   Property

Applies to   ProgressBar, Scroll, Spinner, TrackBar, UpDown

This property is a 2-element vector that specifies the minimum and maximum values of
an object.

List   Object

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

Parents   ActiveXControl, CoolBand, Form, Group, PropertyPage, SubForm, ToolBar, ToolControl

Children   Circle, Cursor, Ellipse, Font, Marker, Poly, Rect, Text, Timer

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

Events   Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, FontCancel, FontOK, GotFocus, Help, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Select

Methods   Animate, ChooseFont, Detach, GetFocus, GetTextSize, ShowSIP

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

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

The Index property specifies or reports the position of Items in the list box as a positive integer value. If Index has the value "n", it means that the "nth" item in Items is displayed on the top line in the list box. However, it is ignored if all the Items fit within the List object. Note that Index can only be set using ⎕*WS* and not by ⎕*WC*. The default value for Index is ⎕*IO*.

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

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

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

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

¯2	scrollbar if required
¯1	scrollbar if required
0	no scrollbar

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

The MultiColumn property is a Boolean value that specifies whether or not the List object displays its items in columns. The default is 0 which produces a single-column display. If MultiColumn is 1, the List object displays its items in columns whose width is defined by the ColumnWidth property.
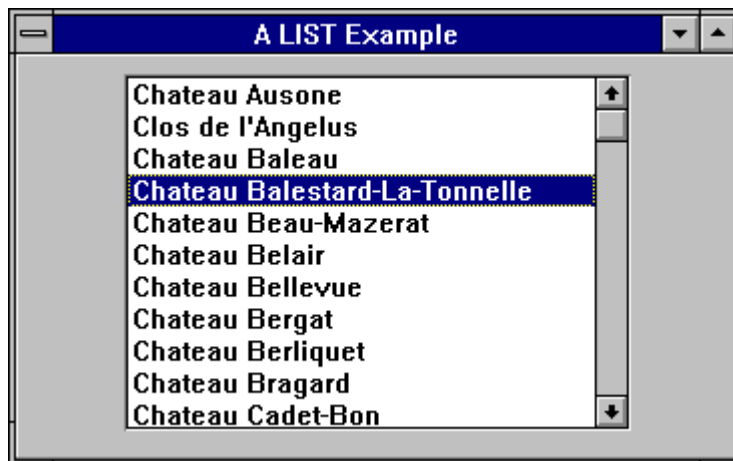
ListTypeLibs   Method 520

Applies to   Root

The ListTypeLibs method reports the names and CLSIDs of all the loaded Type Libraries.

The ListTypeLibs method is niladic.

The result is a nested vector with one element per loaded Type Library.

Each element is a vector of 2-element character vectors. The first is the name of the Type Library; the second is its class identifier or CLSID.

Example:

```
      'EX'⎕WC'OLEClient' 'Excel.Application'
      ⍴ListTypeLibs
3
      ↑⊃ListTypeLibs
Microsoft Excel 9.0 Object Library
{00020813-0000-0000-C000-000000000046}

      ↑⊃¨ListTypeLibs
Microsoft Excel 9.0 Object Library
Microsoft Visual Basic for Applications Extensibility 5.3
Microsoft Office 9.0 Object Library
```

ListView   Object

Purpose	The ListView object displays a collection of items.
Parents	ActiveXControl, CoolBand, Form, Group, PropertyPage, SubForm, ToolBar, ToolControl
Children	Bitmap, Circle, Cursor, Ellipse, Font, Icon, ImageList, Marker, Poly, Rect, Text, Timer
Properties	Type, Items, Posn, Size, Style, Coord, Align, Border, Active, Visible, Event, DragItems, View, AutoArrange, Header, Wrap, EditLabels, ImageListObj, ReportInfo, ColTitles, ImageIndex, VScroll, HScroll, SelItems, Sizeable, Dragable, FontObj, FCol, BCol, CursorObj, AutoConf, Data, Attach, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, ColTitleAlign, ColTitle3D, Translate, Accelerator, AcceptFiles, KeepOnClose, CheckBoxes, FullRowSelect, GridLines, Redraw, MethodList, ChildList, EventList, PropList
Events	BeginEditLabel, Close, ColumnClick, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, EndEditLabel, Expose, FontCancel, FontOK, GotFocus, Help, ItemDblClick, ItemDown, ItemUp, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Select, SetColSize, SetItemPosition
Methods	Animate, ChooseFont, Detach, GetFocus, GetItemPosition, GetItemState, GetTextSize, SetItemState, ShowSIP

The ListView object is a window that displays a collection of items, each item consisting of an icon and a label. The ListView provides several ways of arranging items and displaying individual items. For example, additional information about each item can be displayed in columns to the right of the icon and label. An example of the use of a ListView object is the "My Computer" Windows utility.
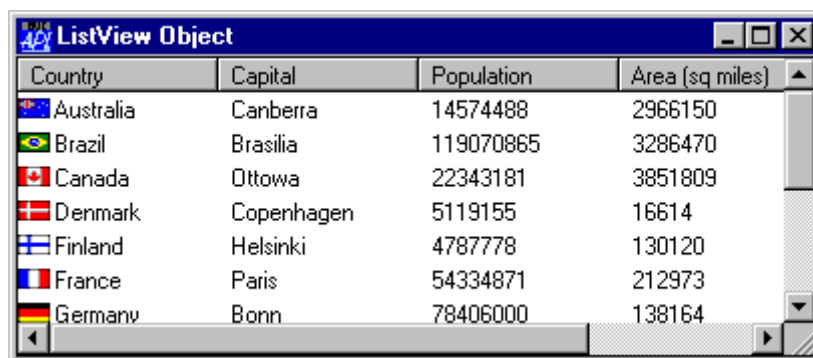
The Items property is a vector of character vectors that specifies the labels for the items displayed by the ListView. The ImageListObj property specifies the names of two ImageList objects that define two sets of icons; a large icon (32x32 pixel) set and a small icon (16x16 pixel) set. Alternatively, ImageListObj may be empty (no icons displayed) or contain just the name of a single large icon ImageList. Finally, the ReportInfo property may contain a matrix of information each row of which is associated with an item.

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

When View is set to '*Report*', the items are displayed in a single column using small icons but with the matrix specified by ReportInfo displayed alongside. In this format, the Boolean Header property determines whether or not the object also provides column headings. Its default value is 1. The column headings themselves are specified by the ColTitles property. Their alignment (and the alignment of the data in the columns beneath them) is defined by the ColTitleAlign property. The appearance of the column titles is further controlled by the ColTitle3D property. This is a Boolean value (default 1) which specifies whether or not the column titles have a 3-dimensional (plinth) appearance. Header and ColTitle3D may only be set when the object is created using ⎕WC and may not subsequently be changed by ⎕WS.

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

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

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

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

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

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

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

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

CheckBoxes, GridLines and FullRowSelect require the Windows Custom Control Library COMCTL32.DLL, Version 4.70 or higher.

LocalAddr   Property

Applies to   TCPSocket

The LocalAddr property is a character vector that specifies the IP address of your computer. Its default value is `'0.0.0.0'` which refers to your default IP address.

Unless your computer has more than one network adapter each identified by a different IP address, you do not need to specify LocalAddr. However, in this case you may use *either* LocalAddr *or* LocalAddrName to identify the adapter. If you specify both properties, the value of LocalAddrName will be ignored.

Note that you may also set the value of LocalAddr to an empty character vector. In this case, the value returned by $\square WG$ will be `'0.0.0.0'`.

LocalAddr may only be specified in the $\square WC$ statement that creates the TCPSocket and may not subsequently be changed using $\square WS$.

LocalAddrName   Property

Applies to   TCPSocket

The LocalAddrName property is a character vector that specifies the host name of your computer. It may be useful when you have more than one network adapter and you wish to avoid hard-coding the IP address.

Note that you may use *either* LocalAddr *or* LocalAddrName to identify the local computer. If you specify both properties, the value of LocalAddrName will be ignored.

LocalAddrName may only be specified by a server TCPSocket. Furthermore, it must be specified in the $\square WC$ statement that creates the TCPSocket object and it may not subsequently be changed using $\square WS$.

When the specified host name has been resolved to an IP address, the TCPSocket will generate a TCPGotAddr event and update the value of LocalAddr accordingly.

For a client TCPSocket, you may not specify LocalAddrName and $\square WG$ returns an empty character vector.

Locale   Property

Applies to   OLEClient

The Locale property specifies the language in which the OLE server, attached to an OLEClient, exposes its methods (functions) and properties (variables).

When you create an OLEClient object, Dyalog APL/W requests the default Type Library associated with the OLE server that you specify. Many OLE servers, such as Excel.Application, provide different names for the methods and properties they expose for different languages. Without Locale, it would be difficult to write an OLE client application that could run in different countries, as the names of the functions and variables may be unpredictable.

Locale is an integer; for example, the value 9 specifies English and the value 12 specifies French.

Locale may only be specified by the *☐WC* statement that is used to create the OLEClient; it may not subsequently be changed using *☐WS*. A table of commonly used Locale values is given below.

Note that Dyadic cannot guarantee that you will actually be given the Locale you specify. This is a function of your specific installation and the OLE server in question. However, Dyadic believes that for Microsoft products, it is a fairly safe bet that the US/English interface will be available in most countries.

Language	Locale
Neutral	0
Danish	6
Dutch	19
English	9
Finnish	11
French	12
German	7
Italian	16
Norwegian	20
Portugese	22
Russian	25
Spanish	10
Swedish	29

LocalPort   Property

Applies to   TCPSocket

The LocalPort property is a scalar integer in the range 0-65535 that identifies the port number associated with a TCPSocket object.

Note that you may use *either* LocalPort *or* LocalPortName to identify the service. The use of LocalPortName is slightly slower but it avoids hard-coding the port number in your program and is generally more flexible. If you specify both properties, the value of LocalPortName will be ignored.

LocalPort may be specified only by the process that is initiating the connection (the server) and must be set by the ⎕WC statement that creates the TCPSocket. LocalPort may not subsequently be changed using ⎕WS.

For a process that is completing a connection, LocalPort is allocated by the system and is effectively read-only.

LocalPortName   Property

Applies to   TCPSocket

The LocalPortName property is a character vector that specifies the port name of the local service that you wish to offer as a server.

Note that you may use *either* LocalPort *or* LocalPortName to identify the service. The use of LocalPortName is slightly slower but it avoids hard-coding the port number in your program and is generally more flexible. If you specify both properties, the value of LocalPortName will be ignored.

LocalPortName may be specified only by the process that is initiating the connection (the server) and must be set by the ⎕WC statement that creates the TCPSocket. LocalPortName may not subsequently be changed using ⎕WS

When the specified port name has been resolved to a port number, the TCPSocket will generate a TCPGotPort event and update the value of LocalPort accordingly.

For a client TCPSocket, you may not specify LocalPortName and ⎕WG returns an empty character vector.

Locator   Object

Purpose	Allows the user to input a point, line or rectangle.
Parents	ActiveXControl, Form, Group, PropertyPage, PropertySheet, Root, Static, SubForm, TCPSocket, ToolBar, ToolControl
Children	Timer
Properties	Type, Posn, Size, LStyle, Style, Coord, Event, Step, Sizeable, CursorObj, Data, Accelerator, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, Locator, Select
Methods	Detach, Wait

This object is used to obtain graphical input from the user. Like a pop-up menu or a MsgBox, the Locator is a *modal* object whose interaction with the user is initiated by a "local" ⎕*DQ*. This is terminated when the user releases a mouse button or presses any key other than a cursor movement key, Shift, Ctrl or Alt. It is usual to initiate the ⎕*DQ* for the Locator from within a callback function attached to a MouseDown (1) Event.

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

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

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

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

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

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

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

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

Locator   Event 80

Applies to   Locator

If enabled, this event is generated when the user releases a mouse button, or presses any key (other than a cursor movement key) during a `⎕DQ` on a Locator object.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 9-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	`'Locator'` or 80
[3] Y:	y-position of Locator after `⎕DQ`
[4] X:	x-position of Locator after `⎕DQ`
[5] H:	height of Locator after `⎕DQ`
[6] W:	width of Locator after `⎕DQ`
[7] Mouse Button:	number of the button which was released (0 if keystroke)
[8] Keystroke:	character scalar or vector containing the "Input Code" for the key that terminated the operation
[9] Shift state:	integer scalar

LockColumns   Method 227

Applies to   Grid

This method is used to lock one or more columns of a Grid object.

The argument to LockColumns is a 1 or 2-element vector as follows.

> [1] Column(s):   integer scalar, vector or matrix

> [2] Lock flag: 0 or 1

Column(s) may be a scalar or a vector specifying the column or columns to be locked or unlocked. Alternatively, it may be a matrix whose first row specifies the columns to be locked and whose second row specifies *where* they are to be locked.

If the *Lock flag* is 1, the corresponding columns are locked. This is the default and may be omitted. If the *Lock flag* is 0, the corresponding columns are unlocked

Examples:

```
F.G.LockColumns 3          ⍝Lock 3rd column
F.G.LockColumns 3 0        ⍝Unlock 3rd column
F.G.LockColumns (4 5)      ⍝Lock 4th & 5th cols
F.G.LockColumns (2 1⍴8 4)  ⍝Lock 8 at 4
```

In all cases, the result is an integer matrix containing the indices of all locked columns and the positions at which they are currently locked.

The expression:

```
F.G.LockColumns ⊂⍬
```

may therefore be used to obtain the indices of the locked columns, and:

```
F.G.LockColumns (F.G.LockColumns ⊂⍬) 0
```

unlocks all currently locked columns.

Locks are additive. If column 4 is locked, locking column 5 results in both columns 4 and 5 being locked.

A locked column remains fixed in position and does not scroll sideways. The user may enter and edit cells in a locked column in the normal way, but the behaviour of the various cell movement keys (Tab, left and right cursor, and so forth) differs when a locked column is encountered. As a general rule, if a keystroke attempts to move the cursor into a locked column from an adjacent column, and the adjacent column has been scrolled, it is unscrolled and the cursor remains in the (new) column adjacent to the fixed column. If not, the cursor moves into the locked column.

When you lock a column, the position you specify for it to be locked at is a position *in the data* and not the physical position of the column as displayed in the Grid. The physical column in the Grid depends upon the value of the Index property at the time it was locked.

If `C` is the value specified for where a given column is to be locked, the value of the physical column `P` at which it will be displayed in the Grid named `GRID` is:

```
P←C-(2⊃GRID ⎕WG 'Index')-⎕IO
```

Furthermore, the position of a locked column given by the result of the LockColumns method changes (with the Index property) as the Grid is scrolled.

LockRows Method 226

Applies to   Grid

This method is used to lock one or more Rows of a Grid object.

The argument to LockRows is a 1 or 2-element vector as follows.

[1] Row(s):	integer scalar, vector or matrix
[2] Lock flag:	0 or 1

Row(s) may be a scalar or a vector specifying the row or rows to be locked or unlocked. Alternatively, it may be a matrix whose first row specifies the data rows to be locked and whose second row specifies *where* in the Grid they are to be locked.

If the *Lock flag* is 1, the corresponding rows are locked. This is the default and may be omitted. If the *Lock flag* is 0, the corresponding rows are unlocked

Examples:

```
F.G.LockRows 3           ⍝Lock 3rd row
F.G.LockRows 3 0         ⍝Unlock 3rd row
F.G.LockRows (4 5)       ⍝Lock 4th and 5th rows
F.G.LockRows (2 1⍴8 4)   ⍝Lock row 8 at 4
```

In all cases, the result is an integer matrix containing the indices of all locked rows and the positions at which they are currently locked.

The expression:

```
F.G.LockRows ⊂θ
```

may therefore be used to obtain the indices of the locked rows, and

```
F.G.LockRows (F.G.LockRows ⊂θ) 0
```

unlocks all currently locked rows.

Locks are additive. If row 4 is locked, locking row 5 results in both rows 4 and 5 being locked.

A locked row remains fixed in position and does not scroll vertically. The user may enter and edit cells in a locked row in the normal way, but the behaviour of the various cell movement keys (Tab, up and down cursor, and so forth) differs when a locked row is encountered. As a general rule, if a keystroke attempts to move the cursor into a locked row from an adjacent row, and the adjacent row has been scrolled, it is unscrolled and the cursor remains in the (new) row adjacent to the fixed row. If not, the cursor moves into the locked row.

When you lock a row, the position you specify for it to be locked at is a position *in the data* and not the physical position of the column as displayed in the Grid.. The physical column in the Grid depends upon the value of the Index property at the time it was locked.

If R is the value specified for where a given row is to be locked, the value of the physical row P at which it will be displayed in the Grid named $GRID$ is given by the expression:

```
P←R-(⊃GRID ⎕WG 'Index')-⎕IO
```

Furthermore, the position of a locked row given by the result of the LockRows method changes (with the Index property) as the Grid is scrolled.

LostFocus   Event 41

Applies to   ActiveXControl, Animation, Button, Calendar, ColorButton, Combo, ComboEx, DateTimePicker, Edit, Form, Grid, Group, List, ListView, MDIClient, ProgressBar, PropertyPage, RichEdit, Scroll, Spinner, SubForm, TrackBar, TreeView

If enabled, this event is generated when the user transfers the keyboard focus away from the object in question.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1] Object:	ref or character vector (object that has lost the focus)
[2] Event code:	`'LostFocus'` or 41
[3] Object:	ref or character vector (object that has received the focus)

If the focus is transferred to a window that is not part of the Dyalog APL GUI Interface, the third element is an empty vector.

The LostFocus event is generated **after** the focus has changed. The default processing is therefore to take no action. However, if you inhibit the event by returning a 0 from your callback function, the focus is automatically restored to the object that had lost it.

`LStyle`   Property

Applies to   Circle, Ellipse, Locator, Poly, Rect

This property determines the type of line used to draw a graphics object. It takes one of the following integer values, or, if the object contains more than one component, a vector of such values.

0	:	solid line
1	:	dashed line
2	:	dotted line
3	:	dash dotted line
4	:	dash dot dotted line
5	:	null line (invisible)

If LStyle is in the range 1-4, the gaps between the dashes and dots are drawn using the colour specified by BCol, or are left undrawn (i.e. transparent) if BCol is not defined.

If LWidth specifies a line width greater than 1 pixel, the value of LStyle is ignored and a solid (thick) line is drawn regardless.

`LWidth`   Property

Applies to   Circle, Ellipse, Poly, Rect

This property determines the width of line used to draw a graphics object. A positive value specifies the line width in pixels. A negative value specifies line width in units of the co-ordinate system defined for the object in the x direction. If the object contains more than one component, LWidth may be a vector.

If LWidth specifies a line width greater than 1 pixel, a solid (thick) line is drawn regardless of the value of LStyle.

MakeGIF   Method 261

Applies to   Bitmap

This method is used to generate an *uncompressed* GIF representation of a picture from a Bitmap object suitable for display by a Web browser.

The MakeGIF method is niladic.

 The result is an integer vector containing the encoded GIF image.

Example

```
      ρGIF←BM.MakeGIF
19620
```

Note: Dyadic has chosen to support uncompressed GIF to avoid the legal complexity of dealing with the licensing issues surrounding the compression algorithm which is patented. If the target Web browser supports PNG (Portable Network Graphics) format, use that instead.

MakePNG   Method 260

Applies to   Bitmap

This method is used to generate a PNG (Portable Network Graphics) representation of a picture from a Bitmap object suitable for display by a Web browser.

The MakePNG is niladic.

 The result of the method is an integer vector containing the encoded PNG image.

Example

```
      ρPNG←BM.MakePNG
4930
```

Note: Although PNG is recognised as the latest graphics standard for displaying pictures, not all Web browsers support it.

MapCols   Property

Applies to   ImageList

The MapCols property specifies whether or not the button colours in bitmaps and icons in an ImageList are re-mapped to reflect the users colour preferences. If your bitmaps and icons represent buttons using the standard windows button colours, this property causes those colours to be changed to suit the user's own colour scheme.

MapCols is a single number with the value 0 (no colour mapping) or 1 (colours are automatically re-mapped. The default is 0.

If MapCols is 1, the following colour mappings are performed:

Colour	Description	Mapped to
0 0 0	Black	Button Text
128 128 128	Dark grey	Button Shadow
191 191 191	Light grey	Button Face
192 192 192	Light grey	Button Face
255 255 255	White	Button Highlight

Marker   Object

Purpose	A graphical object used to draw polymarkers.
Parents	ActiveXControl, Animation, Bitmap, Button, Combo, ComboEx, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, Metafile, Printer, ProgressBar, PropertyPage, PropertySheet, RichEdit, Scroll, Spinner, Static, StatusBar, SubForm, TabBar, TipField, ToolBar, TrackBar, TreeView, UpDown
Children	Timer
Properties	Type, Points, Style, Size, FCol, Coord, Visible, Event, Dragable, OnTop, AutoConf, Data, Accelerator, KeepOnClose, DrawMode, MethodList, ChildList, EventList, PropList
Events	Close, Create, DragDrop, Help, MouseDblClick, MouseDown, MouseMove, MouseUp, Select
Methods	Detach

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

The Style property determines the symbol that is drawn at each of a set of points. Marker styles are specified either by numbers which represent the following symbol shapes or by character vectors containing the names of Bitmap or Icon objects.

0	:	. (the default)
1	:	+
2	:	*
3	:	□
4	:	×
5	:	◇
6	:	○

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

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

Single Set of Polymarkers

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

Style and Size are both simple scalar numbers.

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

First make a Form :

```
'F' ⎕WC 'Form'
```

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

```
'F.M1' ⎕WC 'Marker' (20 10)
```

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

```
'F.M1' ⎕WC 'Marker' (20(10×⍳9))
```

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

```
Y ← 10 10 50 50
X ← 10 50 50 10

'F.M1' ⎕WC 'Marker' (Y X) 1
```

Ditto, but draw them 10% high :

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

Ditto, but use "*" symbols in green :

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

Multiple Sets of Polymarkers

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

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

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

First make a Form :

```
'F' ⎕WC 'Form'
```

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

```
'F.M1' ⎕WC 'Marker'((1 2ρ10 20)(1 2ρ20 20)) (3 5)
```

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

```
Y1 X1 ← (10 10 50 50) (10 50 50 10)
Y2 X2 ← (20 20 40 40) (20 40 40 20)

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

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

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

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

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

Mask   Property

Applies to   Cursor, Icon

This property is used to specify how the bitmap for a Cursor or Icon interacts with the pixels of the screen when it is displayed.

When a Cursor or Icon is displayed, the colour of each pixel occupied by the object on the screen is determined by :

>   a) The colour specified by Bits via CMap
>   b) The value of Mask
>   c) The existing colour of the screen pixel

Mask is a Boolean matrix with the same shape as the Bits property. See Cursor and Icon objects for further details.

MaskCol   Property

Applies to   Bitmap, Form

Specifies the transparent colour for a Bitmap or Form **(Windows 2000 only)**.

MaskCol may be an integer scalar or a 3-element integer vector. If MaskCol is 0 (the default), no transparent colour is defined. If MaskCol is a negative scalar, it specifies a standard Windows colour. See BCol for details.

Otherwise, MaskCol is a 3-element vector of integers in the range 0-255 that specifies the transparent colour in terms of RGB values (the intensity of the red, green and blue components of colour).

For a Bitmap, if MaskCol is non-zero, any pixels specified with the same colour will instead be displayed in whatever colour is underneath the Bitmap. This achieves similar behaviour to that of an Icon.

For a Form (Windows 2000 only), if MaskCol is non-zero, any of the contents of the Form that are specified to be the same colour as MaskCol will be transparent. For example, if MaskCol is 255 0 0 (red), any red items contained in the Form will instead be transparent areas, displaying whatever is behind them on the screen. Mouse events generated over such transparent areas will be passed to any other windows behind them, and will not be reported on the Form itself.

Masked   Property

Applies to   ImageList

The Masked property specifies whether or not the ImageList will contain masked images, i.e. icons. Its default value is 1 which means that the ImageList expects Icons. If you want to use Bitmap objects in an ImageList, you must set Masked to 0. An inappropriate value of Masked will cause the images to be drawn incorrectly.

Masked must be established when the ImageList is created by `⎕WC` and may not subsequently be altered.

MaxButton   Property

Applies to   Form, SubForm

This property determines whether or not a Form or a SubForm has a "maximise" button. Pressing this button will cause a Form to be resized to occupy the entire screen, or a SubForm to occupy the entire area of its parent. Pressing it again will restore the Form or SubForm to its original size. MaxButton is a single number with the value 0 (no maximise button) or 1 (maximise button is provided). The default is 1.

Note that MaxButton is independent of Sizeable, i.e. you can define a Form that can be maximised but not resized. If any of the properties MaxButton, MinButton, SysMenu and Sizeable are set to 1, the Form or SubForm will have a title bar.

MaxDate   Property

Applies to   Calendar, DateTimePicker

The MaxDate property specifies the largest date that the user may select in a Calendar object or in the calendar drop-down of a DateTimePicker.

MaxDate is an IDN value. Its default value is 11249470 which is the maximum date that the Calendar can display.

MaxLength   Property

Applies to   Edit, Spinner

This property specifies the maximum number of characters that the user may enter in a single-line Edit object (Style '*Single*') or Spinner object. Its default value is 0 which implies no limit.

It does not apply to a multi-line Edit object (Style '*Multi*'). MaxLength does not limit the length of the vector that you may assign to the Text property using `⎕WC` or `⎕WS`. However, if you overfill the field in this way, the user must delete excess characters before the object will accept further input.

MaxSelCount   Property

Applies to   Calendar

The MaxSelCount property specifies the maximum number of contiguous days that the user may select in a Calendar object.

MaxSelCount is an integer whose default value is 7.

MaxSelCount is ignored unless the Style property of the Calendar object is set to '*Multi*'.

MDIActive   Property

Applies to   MDIClient

This property contains the name of the SubForm owned by the MDIClient that is currently active. Only one SubForm may be active at a time. You can switch between SubForms in an MDI application under program control by setting this property with `⎕WS`. You can also do this by generating an MDIActivate event.

MDIActivate   Event 42

Applies to   SubForm

This event is generated when the user activates a particular SubForm that is the child of an MDIClient. This occurs when the user clicks the left mouse button in the SubForm or selects it from the menu nominated for this purpose (see MDIMenu property). You may also call MDIActivate as a method.

Note that this event is reported after the action has taken place and cannot be disabled by returning 0 from a callback function or by setting its action code to ¯1.

The event message reported as the result of $\Box DQ$, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	'*MDIActivate*' or 42
[3] Object:	ref or character vector

Note that the 3rd element of the event message is either an empty vector or the ref or name of the SubForm that was previously the active one in the same MDIClient.

MDIArrange   Method 112

Applies to   MDIClient

This method causes the MDIClient object to organise the icons associated with any minimised child Forms into regimented rows and columns. To permit the user to carry out this action, it is recommended that a suitable callback function or expression is attached to a MenuItem or Button. The callback function or expression should then call MDIArrange.

The MDIArrange method is niladic.

MDICascade   Method 110

Applies to   MDIClient

This event causes the MDIClient object to organise its child Forms in an overlapping fashion. To permit the user to carry out this action, it is recommended that a suitable callback function or expression is attached to a MenuItem or Button. The callback function or expression should then invoke the MDICascade method using `⎕NQ`.

The MDICascade method is niladic.

MDIClient   Object

Purpose   Implements Multiple Document Interface (MDI) behaviour.

Parents   ActiveXControl, Form, SubForm

Children   Circle, Ellipse, Font, Marker, Poly, Rect, SubForm, Text, Timer

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

Events   Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, GotFocus, Help, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Select

Methods   Animate, Detach, GetFocus, GetTextSize, MDIArrange, MDICascade, MDITile, ShowSIP

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

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

By default, the MDIClient occupies the entire client area within its parent Form. This is the area within the Form that is not occupied by CoolBars, MenuBars, ToolBars, ToolControls, TabBars, TabControls and StatusBars. In most applications it is therefore not necessary to specify the position and size of the MDIClient object, although you may do so if you want to reserve additional space in the parent Form for other objects. Each of the four sides of an MDIClient object is automatically *attached* to the corresponding side of its parent Form and maintains its position when the parent Form is resized. This means that a default MDIClient always occupies the entire client area of its parent Form, regardless of how the parent is resized.

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

The MDIActive property contains the name of the SubForm that currently has the focus and you can set this with ⎕WS as well as query it with ⎕WG.

You can call methods which cause the MDIClient to organise its child Forms in some way. These methods are:

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

MDIDeactivate Event 43

Applies to   SubForm

This event is generated when the user activates a different SubForm that is the child of an MDIClient, thereby de-activating the current one which causes this event. This occurs when the user clicks the left mouse button in another SubForm or selects it from the menu nominated for this purpose (see MDIMenu property). You may also call MDIDeactivate as a method.

Note that this event is reported after the action has taken place and cannot be disabled by returning 0 from a callback function or by setting its action code to ¯1.

The event message reported as the result of *□DQ*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	*'MDIDeactivate'* or 43
[3] Object:	ref or character vector

Note that the 3rd element of the event message is the ref or name of the SubForm that has now been made the active one in the same MDIClient.

MDIMenu Property

Applies to   MenuBar

This property specifies the name of the Menu object that is nominated as the "Window" menu in an MDI application. If such a menu is defined, the Captions of all the child Forms are automatically added to it below any other Menu or MenuItem objects that the application has created directly. This list is separated from the preceding items by a separator. The entry for the currently active SubForm is checked and the user may switch between SubForms by selecting from this list.

Note that the additional separator and the items representing the list of child forms are not Dyalog APL/W objects and may not be accessed by the application. If you prefer to maintain your own "window list" you should not use this property.

MDITile   Method 111

Applies to   MDIClient

This method causes the MDIClient object to organise its child Forms as a row or column. To permit the user to carry out this action, it is recommended that a suitable callback function or expression is attached to a MenuItem or Button. The callback function or expression should then call the MDITile method.

Note that because there are restrictions concerning the minimum height and width of a window, Windows does not necessarily respond as requested. If the MDIClient is itself of insufficient size, or if it contains a large number of child Forms, Windows may choose to tile the Forms in a row when a column was specified or vice versa. It may also choose to ignore the event entirely.

The argument to MDITile is θ, or a single item as follows:

      [1] Tile Mode: 0 (vertical)

                                   1 (horizontal)

If the argument is θ, the *Tile Mode* defaults to 0.

Menu   Object

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

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

Children   Bitmap, Menu, MenuItem, Separator, Timer

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

Events   Close, Create, DropDown, Select

Methods   Detach, Wait

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

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

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

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

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

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

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

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

MenuBar   Object

Purpose	Specifies a horizontal menu bar displayed at the top of a Form.
Parents	ActiveXControl, Form, SubForm, ToolControl
Children	Bitmap, Menu, MenuItem, Separator, Timer
Properties	Type, Visible, Event, FontObj, Data, EdgeStyle, MDIMenu, Handle, Translate, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create
Methods	Detach

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

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

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

```
'F' □WC 'Form' 'Menu Example'

  'F.M'□WC'MenuBar'

    'F.M.FILE'□WC'Menu' '&File'
    'F.M.MAT' □WC'Menu' '&Materials'

      'F.M.MAT.B'   □WC 'MenuItem' '&Brick'
      'F.M.MAT.C'   □WC 'MenuItem' '&Concrete'
      'F.M.MAT.S'   □WC 'MenuItem' '&Stone'
      'F.M.MAT.SEP' □WC 'Separator'
      'F.M.MAT.W'   □WC 'Menu'     '&Wood'

        'F.M.MAT.W.O' □WC 'MenuItem' '&Oak'
        'F.M.MAT.W.T' □WC 'MenuItem' '&Teak'
        'F.M.MAT.W.M' □WC 'MenuItem' '&Mahogany'
```
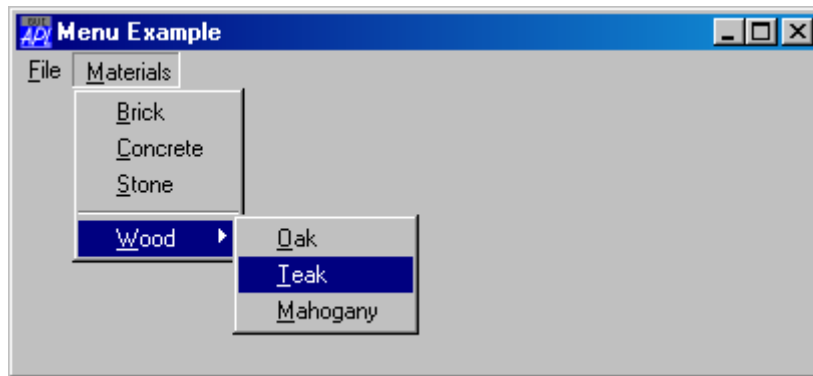
Note that putting a Separator (either Style) in a MenuBar has the effect of breaking the bar vertically, i.e. the next Menu or MenuItem you add will appear on the left-hand side on the line below.

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

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

If a MenuBar is created as the **only** child of a ToolControl object, its menu items are drawn as buttons. Although nothing is done to prevent it, the use of other objects in a ToolControl containing a MenuBar, is not supported.

Under Pocket APL, a MenuBar that is created as a child of a Form, is automatically displayed in the Pocket PC 2002 menu area at the bottom of the screen, rather than along the top edge of the Form.

MenuItem   Object

Purpose	This object allows the user to initiate an action or to select an option from a menu.
Parents	Menu, MenuBar
Children	Bitmap, Timer
Properties	Type, Caption, Posn, Style, Align, Active, Event, Checked, FontObj, FCol, BCol, BtnPix, Data, EdgeStyle, Hint, HintObj, Tip, TipObj, Translate, Accelerator, KeepOnClose, ImageIndex, MethodList, ChildList, EventList, PropList
Events	Close, Create, Select
Methods	Detach

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

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

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

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

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

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

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

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

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

Metafile   Object

Purpose	This object represents a picture in Windows Metafile format.
Parents	ActiveXControl, Bitmap, CoolBand, Form, Group, OLEServer, Printer, PropertyPage, PropertySheet, Root, Static, SubForm, TCPSocket, ToolBar, ToolControl
Children	Circle, Ellipse, Font, Image, Marker, Poly, Rect, Text, Timer
Properties	Type, File, Size, Coord, RealSize, Event, YRange, XRange, Data, Handle, Translate, Accelerator, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, Select
Methods	Detach, FileRead, FileWrite

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

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

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

Whether or not the Metafile object is initialised from a file, you can add graphical components to it by creating child objects. However the Metafile behaves like a Bitmap object in that its children cannot be modified using $\Box WS$ nor can they be removed using $\Box EX$. The components of a Metafile that has been initialised from a .WMF file also cannot be referenced in any way. It is therefore recommended that you use unnamed objects when you create the graphical components of a Metafile. The following statements create an empty Metafile called MF and then draw a line and circle in it.

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

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

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

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

```
'CL' ⎕WC 'Clipboard'
CL.MetafileObj ← 'MF'
```

To save a Metafile object in a file, you call the FileWrite method. The following statements save the Metafile MF in a file called TEST.WMF.

```
MF.File ← 'TEST'
MF.FileWrite
```

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

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

MetafileObj   Property

Applies to   Clipboard

This property is used to copy graphical data to and from the Windows clipboard using the Windows Metafile format.

When you set the MetafileObj property of a Clipboard object to the name of the Metafile object using `⎕WS` its contents are copied to the Windows clipboard in Windows Metafile format.

To import a picture that has been stored in the Windows clipboard in Metafile format you use `⎕WG`. This returns a nested array whose elements correspond to the graphical components of the picture. Each of the elements of the array may be used as the arguments of `⎕WC` to draw the corresponding component of the picture. For example, if the picture stored in C:\MSOFFICE\CLIPART\BIRD.WMF is copied to the Windows clipboard, it may be imported into Dyalog APL/W as follows :

```
        BIRD ← CL.MetafileObj
        ρBIRD
4
```

Each of the items in *BIRD* is a 2-element vector. The first element is a "dummy" object name which you may use or ignore as you wish. The second element is an array that defines a graphical object and is suitable as the right argument of `⎕WC`.

For example :

```
        2⊃4⊃BIRD
POLY  191 397   FSTYLE  0   FILLCOL  0 0 0    ...
      190 402
      187 406
      182 409
      176 410
      172 409
      168 406
      165 402
      164 397
      165 391
      168 387
      172 384
      176 383
      182 384
      187 387
      190 391
      191 397
      189 395
      191 397
```

From this array, you can rebuild the imported picture component by component, either as a Metafile object or directly onto a Form, Static or another object. The following example draws the picture in a Form using the dummy names supplied.

```
'TEST' ⎕WC 'FORM' ('Coord' 'User')
'TEST' ⎕WS ('YRange' 0 1024)('XRange' 0 2048)
TEST.⎕WC/¨BIRD
```

Notice that the co-ordinates of each of the graphical components are typically integers in a co-ordinate system that extends from 0 to 1024 in the y-direction and 0 to 2048 in the x-direction. The simplest way to draw the picture is therefore to set up the same co-ordinate system on a Form as in the example above.

MethodList   Property

Applies to   ActiveXContainer, ActiveXControl, Animation, Bitmap, BrowseBox, Button, Calendar, Circle, Clipboard, ColorButton, Combo, ComboEx, CoolBand, CoolBar, Cursor, DateTimePicker, Edit, Ellipse, FileBox, Font, Form, Grid, Group, Icon, Image, ImageList, Label, List, ListView, Locator, Marker, MDIClient, Menu, MenuBar, MenuItem, Metafile, MsgBox, OCXClass, OLEClient, OLEServer, Poly, Printer, ProgressBar, PropertyPage, PropertySheet, Rect, RichEdit, Root, Scroll, Separator, SM, Spinner, Splitter, Static, StatusBar, StatusField, SubForm, SysTrayItem, TabBar, TabBtn, TabButton, TabControl, TCPSocket, Text, Timer, TipField, ToolBar, ToolButton, ToolControl, TrackBar, TreeView, UpDown

This property reports the names of all the methods supported by a particular COM object or instance of an OLE control. It is a vector of character vectors returned by `⎕WG`. It may not be set using `⎕WC` or `⎕WS`.

MinButton   Property

Applies to   Form, SubForm

This property determines whether or not a Form or SubForm has a "minimise" button. Pressing this button will cause the Form or SubForm to be iconified. Pressing it again will restore the Form to its original size. MinButton is a single number with the value 0 (no minimise button) or 1 (minimise button is provided). The default is 1.

Note that MinButton is independent of Sizeable, i.e. you can define a Form that can be minimised but not resized.

If any of the properties MinButton, MaxButton, SysMenu, and Moveable are set to 1, the Form or SubForm will have a title bar.

MinDate   Property

Applies to   Calendar, DateTimePicker

The MinDate property specifies the smallest date that the user may select in a Calendar or DateTimePicker object.

MinDate is an IDN value. Its default value is -109206 which is the minimum date that the Calendar can display.

MonthDelta   Property

Applies to   Calendar, DateTimePicker

The MonthDelta property specifies the number of months by which a Calendar object scrolls when the user clicks its scroll buttons.

MonthDelta is an integer or an empty vector (zilde). The latter means that the Calendar object scrolls by the number of months that are currently displayed in its window. This is the default.

MouseDblClick   Event 5

Applies to   ActiveXControl, Animation, Button, Calendar, Circle, ColorButton, Combo, ComboEx, DateTimePicker, Edit, Ellipse, Form, Group, Image, Label, List, ListView, Marker, MDIClient, Poly, ProgressBar, PropertyPage, Rect, RichEdit, Scroll, SM, Spinner, Static, StatusBar, StatusField, SubForm, TabBar, TabBtn, Text, ToolBar, ToolButton, ToolControl, TreeView

If enabled, this event is reported when the user presses and then releases a mouse button twice within a short space of time. The duration of this time is set through the Windows Control Panel.

The event message reported as the result of *DQ*, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	*'MouseDblClick'* or 5
[3] Y:	y-position of mouse (number)
[4] X:	x-position of mouse (number)
[5] Button:	button double clicked (number)
	1 = left button
	2 = right button
	4 = middle button
[6] Shift State:	sum of shift key codes (number)
	1 = Shift key is down
	2 = Ctrl key is down

In a graphical object (Circle, Ellipse, Image, Marker, Poly, Rect and Text), the position of the mouse is reported relative to the top-left corner of its bounding rectangle.

Note that double-clicking a mouse button will generate the following sequence of events:

MouseDown
MouseUp
MouseDblClick
MouseUp

MouseDown   Event 1

Applies to   ActiveXControl, Animation, Button, Calendar, Circle, ColorButton, Combo, ComboEx, DateTimePicker, Edit, Ellipse, Form, Group, Image, Label, List, ListView, Marker, MDIClient, Poly, ProgressBar, PropertyPage, Rect, RichEdit, Scroll, SM, Spinner, Static, StatusBar, StatusField, SubForm, TabBar, TabBtn, Text, ToolBar, ToolButton, ToolControl, TrackBar, TreeView, UpDown

If enabled, this event is reported when the user presses one of the mouse buttons. The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1] Object:		ref or character vector
[2] Event code:		`'MouseDown'` or 1
[3] Y:		y-position of mouse (number)
[4] X:		x-position of mouse (number)
[5] Button:		button pressed (number)
		1 = left button
		2 = right button
		4 = middle button
[6] Shift State:		sum of shift key codes (number)
		1 = Shift key is down
		2 = Ctrl key is down

If you enable this event it is advisable that you ALSO enable MouseUp events. Otherwise, the slight delay in running your callback function will cause the down and up sequence to be reversed.

In a graphical object (Circle, Ellipse, Image, Marker, Poly, Rect and Text), the position of the mouse is reported relative to the top-left corner of its bounding rectangle.

MouseEnter   Event 6

Applies to   ActiveXControl, Animation, Button, Calendar, ColorButton, Combo, ComboEx, DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, ProgressBar, PropertyPage, RichEdit, Scroll, SM, Spinner, Static, StatusBar, SubForm, TabBar, ToolBar, ToolControl, TreeView, UpDown

If enabled, this event is reported when the user moves the mouse pointer into (over) an object. The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 3-element vector as follows :

> [1] Object: ref or character vector
>
> [2] Event code :   *'MouseEnter'* or 6
>
> [3] Object: ref or character vector (previous object)

This event is generated when the user moves the mouse pointer across the boundary and into an object. The first element of the event message is the name of the object over which the mouse pointer now resides. The 3rd element of the event message contains the name of the object that was previously under the mouse pointer, or is an empty vector if the mouse pointer was not previously over a Dyalog APL/W object.

MouseLeave   Event 7

Applies to   ActiveXControl, Animation, Button, Calendar, ColorButton, Combo, ComboEx, DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, ProgressBar, PropertyPage, RichEdit, Scroll, SM, Spinner, Static, StatusBar, SubForm, TabBar, ToolBar, ToolControl, TreeView, UpDown

If enabled, this event is reported when the user moves the mouse pointer out of an object. The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 3-element vector as follows :

      [1] Object: ref or character vector

      [2] Event code :   `'MouseLeave'` or 7

      [3] Object: ref or character vector (new object)

This event is generated when the user moves the mouse pointer across the boundary and away from an object. The first element of the event message contains the name of the object that previously contained the mouse pointer and which generated the event when it crossed its boundary. The third element contains the name of the object which now contains the mouse pointer or is an empty vector if the mouse pointer is not now over a Dyalog APL/W object.

MouseMove   Event 3

Applies to   ActiveXControl, Animation, Button, Calendar, Circle, ColorButton, Combo, ComboEx, DateTimePicker, Edit, Ellipse, Form, Group, Image, Label, List, ListView, Marker, MDIClient, Poly, ProgressBar, PropertyPage, Rect, RichEdit, Scroll, Spinner, Static, StatusBar, StatusField, SubForm, TabBar, TabBtn, Text, ToolBar, ToolButton, ToolControl, TrackBar, TreeView, UpDown

If enabled, this event is reported when the user moves the mouse. The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1] Object:	ref or character vector
[2] Event code :	`'MouseMove'` or 3
[3] Y:	y-position of mouse (number)
[4] X:	x-position of mouse (number)
[5] Button:	button released (number)
	1 = left button
	2 = right button
	4 = middle button
[6] Shift State:	sum of shift key codes (number)
	1 = Shift key is down
	2 = Ctrl key is down

In a graphical object (Circle, Ellipse, Image, Marker, Poly, Rect and Text), the position of the mouse is reported relative to the top-left corner of its bounding rectangle.

Note that rapid movement of the mouse will not necessarily cause an overwhelming number of MouseMove events to be reported, as several small movements are automatically combined into one large one.

MouseUp   Event 2

Applies to   ActiveXControl, Animation, Button, Calendar, Circle, ColorButton, Combo, ComboEx, DateTimePicker, Edit, Ellipse, Form, Group, Image, Label, List, ListView, Marker, MDIClient, Poly, ProgressBar, PropertyPage, Rect, RichEdit, Scroll, SM, Spinner, Static, StatusBar, StatusField, SubForm, TabBar, TabBtn, Text, ToolBar, ToolButton, ToolControl, TrackBar, TreeView, UpDown

If enabled, this event is reported when the user releases one of the mouse buttons. The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	*'MouseUp'* or 2
[3] Y:	y-position of mouse (number)
[4] X:	x-position of mouse (number)
[5] Button:	button released (number) 1 = left button 2 = right button 4 = middle button
[6] Shift State:	sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down

In a graphical object (Circle, Ellipse, Image, Marker, Poly, Rect and Text), the position of the mouse is reported relative to the top-left corner of its bounding rectangle.

MouseWheel   Event 8

Applies to   ActiveXControl, Animation, Button, Calendar, ColorButton, Combo, ComboEx, DateTimePicker, Edit, Form, Group, Label, List, ListView, MDIClient, ProgressBar, PropertyPage, RichEdit, Scroll, Spinner, Static, StatusBar, SubForm, TabBar, ToolBar, ToolControl, TreeView

If enabled, this event is reported when the user rotates the mouse wheel.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 9-element vector as follows :

[1] Object:		ref or character vector
[2] Event name or code:		`'MouseWheel'` or 8
[3] Y:		y-position of mouse (number)
[4] X:		x-position of mouse (number)
[5] Button:		button pressed 1 = left button 2 = right button 4 = middle button
[6] Shift State:		sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down
[7] Delta:		integer
[8] Lines:		integer
[9] Wheel Delta:		integer

The value of *Delta* indicates the distance that the wheel is rotated, expressed in multiples or divisions of *Wheel Delta*. A positive value indicates that the wheel was rotated forward, away from the user; a negative value indicates that the wheel was rotated backward, toward the user.

Lines specifies the number of lines to scroll when the wheel is rotated by I *Mouse Delta* unit. A value of ¯1 indicates that that a whole screen is to be scrolled. These values are defined by the user's preferences (*Control Panel/Mouse*)

Moveable   Property

Applies to   Form, SubForm

This property determines whether or not a Form or SubForm can be moved by the user. It is a single number with the value 0 (Form cannot be moved) or 1 (Form is moveable). If any of the properties MinButton, MaxButton, SysMenu, and Moveable are set to 1, the Form or SubForm will have a title bar. However, a Form or SubForm with a title bar is not necessarily moveable.

MsgBox   Object

Purpose	Provides a "modal" dialog box for displaying messages, errors, warnings and other information. The dialog box has a title, one or more lines of text, and up to three buttons.
Parents	ActiveXControl, Calendar, CoolBand, DateTimePicker, Form, Grid, OLEServer, PropertyPage, PropertySheet, Root, SubForm, TCPSocket, ToolBar, ToolControl
Children	Timer
Properties	Type, Caption, Text, Style, Btns, Default, Event, Data, EdgeStyle, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, MsgBtn1, MsgBtn2, MsgBtn3
Methods	Detach, Wait

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

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

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

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

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

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

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

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

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

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

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

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

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

```
Caption←'Default MsgBox' ◇ Text←'Hello World'
'Msg' ⎕WC 'MsgBox' Caption Text ◇ ⎕DQ 'Msg'
```
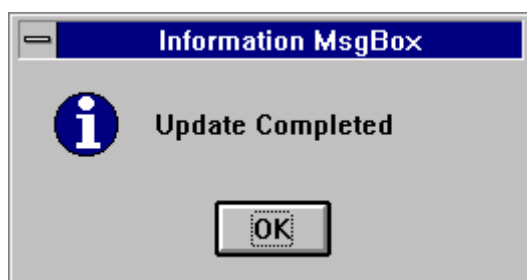


```
Caption←'Information MsgBox' ◇ Text←'Update Completed'
'Msg' ⎕WC 'MsgBox' Caption Text 'Info' ◇ ⎕DQ 'Msg'
```
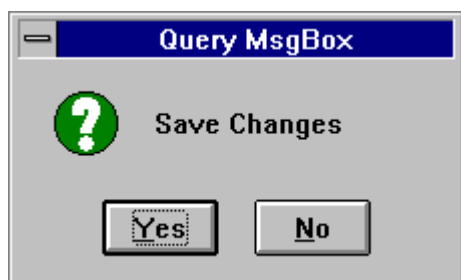


```
Caption←'Query MsgBox' ◇ Text←'Save Changes'
'Msg' ⎕WC 'MsgBox' Caption Text 'Query' ◇ ⎕DQ 'Msg'
```
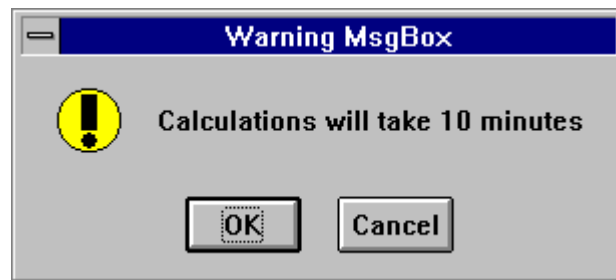
```
Caption←'Warning MsgBox'
Text←'Calculations will take 10 minutes'
'Msg' ⎕WC 'MsgBox' Caption Text 'Warn' ◊ ⎕DQ 'Msg'
```
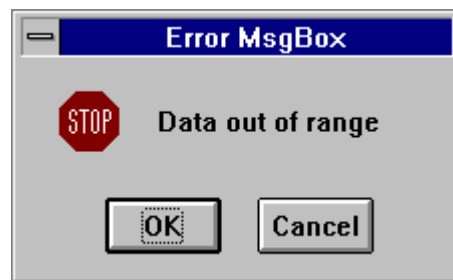


```
Caption←'Error MsgBox'
Text←'Data out of range'
'Msg' ⎕WC 'MsgBox' Caption Text 'Error' ◊ ⎕DQ 'Msg'
```



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

MsgBtn1   Event 61

Applies to   MsgBox

If enabled, this event is reported when the user responds to a MsgBox object by clicking its first (leftmost) button. The event message reported as the result of □*DQ*, or supplied as the right argument to your callback function, is a 2-element vector as follows:

[1] Object: ref or character vector

[2] Event code: '*MsgBtn1*' or 61

MsgBtn2   Event 62

Applies to   MsgBox

If enabled, this event is reported when the user responds to a MsgBox object by clicking its second (from the left) button. The event message reported as the result of □*DQ*, or supplied as the right argument to your callback function, is a 2-element vector as follows:

[1] Object: ref or character vector

[2] Event code: '*MsgBtn2*' or 62

MsgBtn3   Event 63

Applies to   MsgBox

If enabled, this event is reported when the user responds to a MsgBox object by clicking its third (from the left) button. The event message reported as the result of □*DQ*, or supplied as the right argument to your callback function, is a 2-element vector as follows:

[1] Object: ref or character vector

[2] Event code: '*MsgBtn3*' or 63

MultiColumn   Property

Applies to   List

MultiColumn is Boolean and specifies whether or not a List object displays its items in a
single column (0, the default) or in multiple columns (1). MultiColumn may only be set
by `⎕WC` and cannot be changed using `⎕WS` after the object has been created. Note that a
MultiColumn List will use the minimum number of columns that are required to make
the items fit within it and will reconfigure itself automatically when resized. The
following example illustrates its use.

```
'F'⎕WC'Form' 'MultiColumn List'('Size' 23 32)
'F.L'⎕WC'LIST' AIRPORTS (0 0)(100 100)('MultiColumn' 1)
```
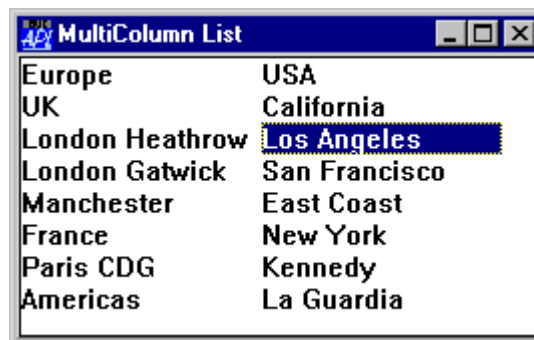

See also: ColumnWidth

MultiLine   Property

Applies to   TabControl, ToolControl

The MultiLine property determines whether or not the tabs or buttons will be arranged
in multiple flights or multiple rows/columns in a TabControl or ToolControl object.

MultiLine is a single number with the value 0 (single flight of tabs, or single row/column
of buttons) or 1 (multiple flights of tabs or multiple rows/columns of buttons); the
default is 0.

If MultiLine is 0 and there are more tabs or buttons than will fit in the space provided,
the TabControl displays an UpDown which allows the user to scroll.

However, If MultiLine is 0 in a ToolControl, the buttons are clipped, and the user may have to resize the object to see them all.

See also: Justify, TabSize.

MultiSelect   Property

Applies to   TabControl

The MultiSelect property specifies whether or not the user can select more than one button in a TabControl at the same time, by holding down the Ctrl key when clicking.

MultiSelect is a single number with the value 0 (only 1 button may be selected) or 1 (more than one button may be selected); the default is 0.

MultiSelect apples only if the Style of the TabControl is *'Buttons'* or *'FlatButtons'*, and is ignored if Style is *'Tabs'*.

Note that the State property of the associated TabButton object reports whether or not the button is selected.

NameFromHandle   Method 136

Applies to   Root

This method is used to obtain the name of a particular object from the value of its Handle property.

The argument to NameFromHandle is a single item as follows:

> [1] Handle: The value of the Handle property from an existing object.

The result is a character vector containing the name of the object. If the Handle does not belong to an object, the result is an empty vector.

NetClient Object

Purpose	The NetClient object represents an instance of a Microsoft .Net class.
Parents	NetClient, NetType, Root
Children	NetClient, Timer
Properties	(None)
Events	(None)
Methods	(None)

The NetClient object represents an instance of a .Net class.

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

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

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

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

NetType Object

Purpose	The NetType object is used to export a namespace as a Microsoft.Net class.
Parents	NetType, Root
Children	Bitmap, NetClient, NetType, TCPSocket, Timer
Properties	BaseClass
Events	Close, Create
Methods	(None)

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

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

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

NewLine   Property

Applies to   CoolBand

The NewLine property specifies whether or not a CoolBand occupies the same row as an existing CoolBand, or is displayed on a new line within its CoolBar parent.

NewLine is a single number with the value 0 (same row) or 1 (new row); the default is 1.

The value of NewLine in the first CoolBand in a CoolBar is always 1, even if you specify it to be 0.

When the user drags a CoolBand to another row, the value of its NewLine property, and that of any other CoolBand affected by the move, will change.

You may move a CoolBand to the previous or next row by changing its NewLine property (using ⎕WS)from 1 to 0, or from 0 to 1 respectively.

NewPage   Method 102

Applies to   Printer

This method causes a Printer to start a new page.

The NewPage method is niladic.

 If you attach a callback function to this event and have it return a value of 0, the page throw will not occur.

OCXClass Object

Purpose	This object provides access to OLE (ActiveX) Controls.
Parents	ActiveXControl, CoolBand, Form, Grid, OLEServer, PropertyPage, Root, SubForm, TCPSocket, ToolBar, ToolControl
Children	(None)
Properties	Type, ClassName, Event, Data, Translate, ClassID, KeepOnClose, TypeList, HelpFile, ToolboxBitmap, LicenseKey, QueueEvents, MethodList, ChildList, EventList, PropList
Events	(None)
Methods	Browse, Detach, GetEventInfo, GetMethodInfo, GetPropertyInfo, GetTypeInfo, OLEAddEventSink, OLEDeleteEventSink, OLEListEventSinks, SetMethodInfo, SetPropertyInfo, ShowHelp, ShowProperties

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

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

OKButton   Property

Applies to   Form

OKButton applies only to Pocket APL. In versions of Dyalog APL for other platforms, it has no effect.

This is a Boolean property that specifies whether or not an [OK] button appears in the title bar of a Form. Its default value is 0.

OKButton may only be specified when the Form is created using $\square WC$; you cannot subsequently change its value.

If OKButton is 1, the Form displays an [OK] button in its title bar in place of the standard [X] button.

When the user clicks the [OK] button, the system will press the default button, which is specified by the Default property of a Button on the Form.

If there is no default button, the Form will generate a Close event.

OLEAddEventSink   Method 540

Applies to   OCXClass, OLEClient

This method connects a named event sink to a COM object and adds the events defined by that event sink to the EventList property of the associated namespace.

The argument to OLEAddEventSink is a single item as follows:

      [1] Event sink name: character vector

The result is a number that represents the handle of the event sink. This may be subsequently required.

OLEClient   Object

Purpose	The OLEClient object provides access to an OLE Automation Server
Parents	ActiveXControl, CoolBand, Form, OLEClient, OLEServer, Root, TCPSocket
Children	Form, OLEClient, TCPSocket, Timer
Properties	Type, ClassName, Event, Data, Handle, ClassID, KeepOnClose, TypeList, HelpFile, LastError, Locale, AutoBrowse, QueueEvents, InstanceMode, MethodList, ChildList, EventList, PropList
Events	(None)
Methods	Browse, Detach, GetEventInfo, GetMethodInfo, GetPropertyInfo, GetTypeInfo, OLEAddEventSink, OLEDeleteEventSink, OLEListEventSinks, OLEQueryInterface, SetMethodInfo, SetPropertyInfo, ShowHelp

The OLEClient object allows you to control OLE Servers, which may be written in a variety of different programming languages, including Dyalog APL itself.

The ClassName property specifies the name of the OLE object to which the new object named by the left argument of ⎕WC is to be connected. A list of all the OLE Server objects installed on your system may be obtained from the OLEServers property of Root. ClassName may only be specified by ⎕WC.

Alternatively, the OLE object may be identified by the ClassID property.

The AutoBrowse property and Browse method are no longer relevant and are ignored. They are retained only for backwards compatibility with previous versions of Dyalog APL.

Note that the PropList and MethodList properties of an OLEClient instance contain the names of the properties and methods exposed by the corresponding OLE Object in addition to the generic properties and methods of the OLEClient class.

If you call an OLE method with an invalid parameter, set a read-only property, or assign it an invalid value, the LastError property of the OLEClient and Root objects will contain error information generated by OLE.

OLEControls   Property

Applies to   Root

The OLEControls property reports a list of the OLE Controls installed on your computer. This information is obtained from the Windows registry. Its value is a nested vector with one element per OLE Control. Each element is a vector of 2-element character vectors. The first is the name of the OLE Control; the second is its class identifier. The latter is a string of hexadecimal characters that uniquely identifies the Control.

OLEDeleteEventSink   Method 541

Applies to   OCXClass, OLEClient

This method disconnects a named event sink from a COM object and removes the events defined by that event sink from the EventList property of the associated namespace.

This method may be used to remove an event sink that was established automatically when the OLE object was created.

The argument to OLEDeleteEventSink is a single item as follows:

      [1] Event sink name: character vector

OLEListEventSinks   Method 542

Applies to   OCXClass, OLEClient

This method returns the names of *event sinks* that are currently connected to a COM object.

The list contains the names of all the event sinks that were connected automatically when the object was created, together with any that you have added subsequently using OLEAddEventSink.

The OLEListEventSinks method is niladic.

The result is a vector of character vectors containing the names of the event sinks connected to the object.

OLEQueryInterface Method 543

Applies to ActiveXContainer, OLEClient

This method is used to obtain the methods and properties associated with a particular *interface* that is provided by a COM object. An interface is simply a pointer to a table of methods (not properties) that are exported by an object.

Note that methods and properties exported using the standard IDispatch interface are established automatically when the object is created. OLEQueryInterface is required only to support alternative or additional interfaces that the object may implement.

The argument to OLEQueryInterface is a single item as follows:

[1] Interface name: character vector

The result is a namespace.

It is normal, although not strictly required, that the new namespace be a child of the one for which the method is run.

Note that if the object does not support a type library, the new namespace will be empty and you will have to establish functions corresponding to the methods exported by the interface using SetMethodInfo.

OLERegister Method 530

Applies to OLEServer

This method is used to register an OLEServer object. This method may be used to install Dyalog APL OLE Servers as part of a run-time installation.

The OLERegister method is niladic.

OLEServer   Object

Purpose	The OLEServer object is used to establish a namespace as an OLE Server object that can be used by an OLE Automation client.
Parents	ActiveXControl, Form, OLEServer, Root
Children	Bitmap, BrowseBox, Clipboard, Cursor, FileBox, Font, Form, Icon, ImageList, Menu, Metafile, MsgBox, OCXClass, OLEClient, OLEServer, Printer, PropertySheet, TCPSocket, Timer, TipField
Properties	Type, ClassName, Event, Data, Handle, ExportedFns, ExportedVars, ClassID, KeepOnClose, TypeLibID, TypeLibFile, ServerVersion, LastError, RunMode, ShowSession, MethodList, ChildList, EventList, PropList
Events	Close, Create
Methods	Detach, OLERegister, OLEUnregister, SetEventInfo, SetFnInfo, SetVarInfo

The OLEServer object allows you to export an APL namespace so that its functions and variables become directly accessible to an OLE Automation client application such as Microsoft Visual Basic or Microsoft Excel.

An OLEServer may be saved as an *out-of-process* OLE server (in a workspace) or as an *in-process* OLE server (in a DLL). See *Interface Guide* for details.

When you create an OLEServer object, APL allocates various OLE attributes to it. For example, the CLSID, which uniquely identifies the object, is assigned at this stage. However, the object is not actually *registered* until you execute $)SAVE$ or run the OLERegister method.

Registration involves updating the Windows registry with information about the object itself, such as its name, the command line required to start it, and so forth. Registration also records information about all of the functions and variables that your object exposes. Registration is therefore a non-trivial operation and should be delayed until the point when you are ready to test your OLEServer.

You may create an empty OLEServer object and then define functions and variables within it. Alternatively, you may convert an existing namespace which is already populated with functions and variables. The latter method is recommended as it implies less registry activity during the development of the object.

The ExportedFns and ExportedVars properties specify the names of the functions and variables that will be exposed by the object to OLE clients.

The RunMode property is a character vector that specifies how the object serves multiple clients. It may be `'MultiUse'` (the default), `'SingleUse'`, or `'RunningObject'`.

The ShowSession property is either 0 (the default) or 1 and specifies whether or not the APL Session window is displayed when the first instance of the OLEServer is created.

RunMode and ShowSession apply only to *out-of-process* OLEServers.

OLEServers   Property

Applies to   Root

The OLEServers property is a read-only property that reports the names and CLSIDs of all the OLE Automation servers installed on your computer. This information comes from the Windows registry.

Its value is a nested vector with one element per OLE Server.

Each element is a vector of 2-element character vectors. The first is the name of the OLE Server; the second is its class identifier or CLSID.

OLEUnregister   Method 531

Applies to   OLEServer

This method is used to unregister an OLEServer object that has previously been saved by Dyalog APL.

The OLEUnregister method is niladic.

This method removes all traces of the object from the Windows registry and erases its Type Library file.

Note that the name of the object removed from the registry is the name of the OLEServer object prefixed by the string "dyalog."

OnTop Property

Applies to Circle, Ellipse, Form, Image, Marker, Poly, PropertySheet, Rect, SubForm, TabBar, Text, ToolBar

This property may be used to cause a Form or SubForm to be displayed on top of all other windows, even those owned by other applications. It is also used to specify how graphical objects are drawn in a Grid.

Normally, Forms are brought to the front when they receive the input focus. Forms that do not have the input focus may be partially obscured by the one that does. If OnTop is set to 1, the Form or SubForm remains at the front even if it doesn't have the input focus. Indeed, it may partially obscure the Form that does have the focus. The default value is 0 (normal).

More than one Form may have OnTop set to 1. If so, these Forms appear on top of all others, but may overlap one another. Other applications may also have windows with this attribute.

For a graphical object, the OnTop property controls how it is drawn in a Grid relative to the grid lines and cell text. OnTop is applicable only if the graphic is the child of a Grid and is otherwise ignored.

0   Graphical object is drawn behind grid lines and cell text
1   Graphical object is drawn on top of grid lines but behind cell text
2   Graphical object is drawn on top of grid lines and cell text

Orientation   Property

Applies to   Printer

The Orientation property specifies the orientation of the paper on a Printer object. It is a simple character vector which is either *'Portrait'* or *'Landscape'*. When you create a Printer object, the default value of the Orientation property is determined by the current setting for the corresponding printer device.

The effect of changing Orientation using *⎕WS* is to spool the current page (effectively the same as sending a NewPage event) and then to change the orientation of the paper. Note that the values of the first 2 elements of the DevCaps property change accordingly.

You may also set Orientation when you create the Printer object with *⎕WC*.

In neither case does the global setting for the printer device change.

OtherButton   Property

Applies to   ColorButton

The OtherButton property is a Boolean value that specifies whether or not the colour selection drop-down on a ColorButton object has a button that allows the user to bring up the Windows colour selection dialog box from which any available colour may be selected.

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

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

Note that the Pocket PC 2002 colour selection dialog box does not provide the facility to select *custom colours*, so this functionality is not available in Pocket APL.

OverflowChar   Property

Applies to   Grid

The OverflowChar property specifies the character to be displayed in place of the digits when a numeric value cannot be displayed in its entirety in a Grid cell. If the value of OverflowChar is an empty vector (the default) the data in a numeric cell is simply clipped if it is too wide to fit in the cell. For example:

```
'F'⎕WC'Form'('Coord' 'Pixel')('Size' 101 296)
F.Caption' ← 'OverflowChar Property'
DATA←3 3ρ12 123456789 13 9876543 99 456 10 99 1236.893
'F.G'⎕WC'Grid'DATA(0 0)(101 296)
F.G.CellWidths ← 65
F.G.OverflowChar ← '#'
```

OverflowChar Property	A	B	C
1	12	#######	13
2	#######	99	456
3	10	99	####.##i

The same Grid without OverflowChar being defined appears as follows. Notice how the numbers have been truncated

OverflowChar NOT specified	A	B	C
1	12	3456789	13
2	9876543	99	456
3	10	99	236.893

PageActivate   Event 360

Applies to   PropertyPage

If enabled, this event is reported when the user switches from one PropertyPage to another in a PropertySheet object. This event is reported by the new page *after* the page change has occurred and the page change may not be disabled by a callback function.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 2-element vector as follows :

      [1] Object: ref or character vector

      [2] Event name or code: *'PageActivate'* or 360

You may select a particular page by calling PageActivate as a method, or by setting the PageActive property of the PropertySheet.

PageActive   Property

Applies to   PropertySheet

The PageActive property specifies the name of the current PropertyPage in a PropertySheet. You may select a particular page by setting this property using $\square WS$ or by generating a PageActivate event.

PageApply   Event 350

Applies to   PropertyPage

If enabled, this event is reported when the user clicks the Apply button in a PropertySheet. Note however, that the event is actually reported by each of its PropertyPage objects whose Changed property is currently 1, i.e. the event is reported by each of the pages that the user has changed.

The default processing for this event is to set the Changed property of the PropertyPage to 0. If you disable the event or return a 0 from a callback function, the Changed property is not reset. Note that the Apply button in a PropertySheet is active if the value of the Changed property of *any* of the PropertyPage objects is 1.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1] Object: ref or character vector

[2] Event name or code: '*PageApply*' or 350

PageBack   Event 353

Applies to   PropertyPage

If enabled, this event is reported when the user switches from one PropertyPage to another in a Wizard PropertySheet object by clicking its Back button. This event is reported by the old page *after* the page change has occurred and the page change may not be disabled by a callback function.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1] Object: ref or character vector

[2] Event name or code: '*PageBack*' or 353

PageCancel   Event 351

Applies to   PropertyPage

If enabled, this event is reported when the user presses the Cancel button in a PropertySheet object and is reported by the current PropertyPage. This event is reported for information only and may not be disabled by a callback function. However, the operation will also generate a Close event reported by the PropertySheet itself that may be disabled by a callback.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1] Object: ref or character vector

[2] Event name or code: '*PageCancel*' or 351

PageChanged   Event 356

Applies to   PropertyPage

If enabled, this event is reported when the Changed property of a PropertyPage is altered by user action. It is *not* reported if you reset the Changed property using `⎕WS`.

The Changed property is reset by two separate user actions. It is set to 1 when the user alters any of the controls on the PropertyPage. It is reset to 0 when the user clicks the Apply button, although this action may be disabled by a callback function on the PageApply event.

The PageChanged event is reported for information only and may not itself be disabled or affected by a callback function.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 3-element vector as follows :

     [1] Object: ref or character vector

     [2] Event name or code: `'PageChanged'` or 356

     [3] Changed value: New value for Changed property (0 or 1).

PageDeactivate   Event 361

Applies to   PropertyPage

If enabled, this event is reported when the user switches from one PropertyPage to another in a PropertySheet object. This event is reported by the old page *after* the page change has occurred and the page change may not be disabled by a callback function.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

     [1] Object: ref or character vector

     [2] Event name or code: `'PageDeactivate'` or 361

PageFinish Event 355

Applies to PropertyPage

If enabled, this event is reported when the user clicks the Finish button in a Wizard PropertySheet. This event is reported by current (last) PropertyPage. The event is reported for information only and cannot be affected by a callback function.

The event message reported as the result of ⎕*DQ*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

     [1] Object: ref or character vector

     [2] Event name or code: *'PageFinish'* or 355

PageHelp Event 352

Applies to PropertyPage

If enabled, this event is reported when the user clicks the Help button in a Wizard PropertySheet. This event is reported by current PropertyPage. The event is reported for information only and cannot be affected by a callback function.

The event message reported as the result of ⎕*DQ*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

     [1] Object: ref or character vector

     [2] Event name or code: *'PageHelp'* or 352

PageNext   Event 354

Applies to   PropertyPage

If enabled, this event is reported when the user switches from one PropertyPage to another in a Wizard PropertySheet object by clicking its Next button. This event is reported by the old page *after* the page change has occurred and the page change may not be disabled by a callback function.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 2-element vector as follows :

> [1] Object: ref or character vector

> [2] Event name or code: '*PageNext*' or 354

PageWidth   Property

Applies to   RichEdit

The PageWidth property specifies the width of the page in a RichEdit object and is the dimension that is used to apply text wrapping and paragraph formatting to the text in the object. PageWidth is a single integer value specified in Twips. The default value of PageWidth is that defined for the default printer. If there are no printers installed, the default value is 0 which disables text wrapping. You may find it convenient to set PageWidth to the width of the RichEdit window or to a value that is appropriate for your printer.

PaperSize   Property

Applies to   Printer

The PaperSize property specifies the size of paper to be used for printing

PaperSize may be a character vector containing the name of the paper size (e.g.
`'Legal 8 1/2 x 14 in'` or `'A4 210 x 297 mm'`) or a 2-element integer
vector that specifies the desired height and width of the paper in tenths of a millimetre
(e.g. 3556 2159 or 2970 2099).

The default value of PaperSize is the name of the paper size associated with the current
printer settings.

You can obtain a list of supported paper sizes from the PaperSizes property.

PaperSizes   Property

Applies to   Printer

The PaperSizes property is a read-only property that provides the names and dimensions
of the various different paper sizes supported by the printer associated with the Printer
object.

PaperSizes is a nested vector of 2-element vectors which contain the name, and height
and width of each paper size respectively. Dimensions are reported in tenths of a
millimetre.

You may set or query the current paper size using the PaperSize property.

PaperSource   Property

Applies to   Printer

The PaperSource property is a character vector that specifies the name of the paper bin
to be used as the paper source for printing.

An empty character vector (the default) means the default bin, Otherwise, PaperSource
must be a member of the PaperSources property.

PaperSources   Property

Applies to   Printer

The PaperSources property is a read-only property that provides the names of the paper bins installed on the printer associated with the Printer object. It is a vector of character vectors.

You may select which of the bins is to be used by specifying the PaperSource property.

ParaFormat   Property

Applies to   RichEdit

The ParaFormat property describes the current paragraph format or the paragraph format of the currently selected text in a RichEdit object. It is a 6-element nested array structured as follows:

[1]   A character vector that specifies the text alignment. This may be `'Left'` (the default), `'Right'` or `'Centre'`.

[2]   The size of the indentation of the first line in the paragraph measured from the left margin in Twips.

[3]   The size of the horizontal offset of the start of the second and subsequent lines. This is measured in Twips relative to the first line indentation specified in element [2].

[4]   The size of the right indentation measured in Twips from the right margin.

[5]   An integer value specifying the bullet/numbering option. 0 means no numbering, 1 means bullets.

[6]   An integer vector specifying the size of any tab stops measured in Twips from the left margin and specified in ascending order.

If there is no text selected, ParaFormat specifies the *current* paragraph formatting format, i.e. that which will be used to format the current (and subsequent) lines of characters that the user enters. If there *is* text selected ParaFormat specifies the paragraph formatting of the selected block of text. If the format is not strictly homogeneous, ⎕*WG* will report the format of the first paragraph in the selected block

Setting ParaFormat will set the format of the currently selected block of text. To set the format of an arbitrary block of text you must select it first using SelText.

Password   Property

Applies to   Edit, Spinner

This property specifies the character that is echoed when a user enters data into a single-line Edit object (Style *'Single'*) or a Spinner. It does not apply to a multi-line object (Style *'Multi'*. If Password is empty (the default) the character echoed is the same as the character the user entered. If Password is set to (say) the asterisk character (*), the object will display asterisks as the user types into it.

PathWordBreak   Property

Applies to   ComboEx

If set, the edit control portion of the ComboEx will use the forward slash (/), back slash (\), and period (.) characters as word delimiters. This makes keyboard shortcuts for word-by-word cursor movement (Ctrl + arrow keys) effective in path names and URLs.

This property is ignored under Windows 95/98.

Picture   Property

Applies to   ActiveXControl, Button, Clipboard, CoolBand, Form, Group, Image, MDIClient, SM, Static, StatusBar, StatusField, SubForm, TabBar, ToolBar

For ActiveXControl, Button, Form, Group, MDIClient, Static, StatusBar, StatusField, SubForm, SM, TabBar or ToolBar, this property defines a Bitmap, Icon, or Metafile which is drawn as a *background* on the object. Other controls and graphical objects are drawn on top of this background.

When it refers to a Metafile, the Picture property is a simple character vector that specifies the name of the Metafile to be drawn in the object. When it refers to a Bitmap or Icon, the value of the Picture property is a 2-element nested vector whose elements specify the name of the Bitmap, or Icon, and the manner in which it is displayed. This is specified as an integer as follows:

0   the Bitmap or Icon is drawn in the top left corner of the object.

1   the Bitmap or Icon is tiled (replicated) to fill the object.

2   the Bitmap is scaled (up or down) to fit exactly in the object. This setting
         does not apply to an Icon whose size is fixed.

3   the Bitmap or Icon is drawn in the centre of the object. This is the default.
         Note that the centre of the Bitmap is positioned over the centre of the object,
         so that you see the middle portion of a Bitmap that is larger than the object in
         which it is displayed.

For example, the following statements produce a Form filled with the *CARS* bitmap.

```
'CARS' ⎕WC 'Bitmap' 'C:\WINDOWS\CARS'
'f1' ⎕WC 'Form' ('Picture' 'CARS' 1)
```

An easy way to provide a customised tool button is to create a Button whose Picture
property specifies the name of a Bitmap or Icon, using *display manner* 3 (the default).
This causes the corresponding bitmap or icon to be drawn in the centre of the Button. So
long as the Button is larger than the bitmap or icon, its borders (which give it its 3-
dimensional appearance and "pushbutton" behaviour) will be unaffected. Note that if
Picture is set on a Button whose Style is '*Radio*' or '*Check*', the Button assumes
pushbutton appearance, although its radio/check behaviour is preserved.

For an Image object, the Picture property is a simple character vector specifying the
name of a Bitmap, Icon or Metafile object to be drawn, or a vector of names. The Image
is a graphical object and is drawn *on top of* the background.

PName Property

Applies to   Font, Printer

This property is a character vector that specifies the face name for a Font object, or the printing device associated with a Printer. It is case-independent.

For a Printer, PName contains the description of the printer followed by a comma (,) and then the device to which it is attached. The device ends with a colon (:).

Example

```
      'PR1' □WC 'Printer'

      PR1.PName
IBM 4039 LaserPrinter PS,LPT2:
```

Points Property

Applies to   Circle, Ellipse, Image, Marker, Poly, Rect, Text

This property specifies the co-ordinates for a graphics object. It may define a single set of co-ordinates, or be a nested vector containing several sets of co-ordinates.

Each set of co-ordinates may be:

a)   a 2-column numeric matrix containing y-values in column 1 and x-values in column 2, or

b)   a 2-element numeric vector whose first element specifies y-values and whose second element specifies x-values. The two elements must be of equal length unless one or both is a scalar in which case scalar extension applies.

For further details, see the specifications for the relevant objects.

Poly   Object

Purpose	A graphical object used to draw lines, polygons, and filled areas.
Parents	ActiveXControl, Animation, Bitmap, Button, Combo, ComboEx, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, Metafile, Printer, ProgressBar, PropertyPage, PropertySheet, RichEdit, Scroll, Spinner, Static, StatusBar, SubForm, TabBar, TipField, ToolBar, TrackBar, TreeView, UpDown
Children	Timer
Properties	Type, Points, FCol, BCol, LStyle, LWidth, FStyle, FillCol, Coord, Visible, Event, Dragable, OnTop, CursorObj, AutoConf, Data, Accelerator, KeepOnClose, DrawMode, MethodList, ChildList, EventList, PropList
Events	Close, Create, DragDrop, Help, MouseDblClick, MouseDown, MouseMove, MouseUp, Select
Methods	Detach

The Points property specifies one or more sets of co-ordinates through which one or more lines are drawn. The resulting polygon(s) may also be filled.

LStyle and LWidth define the style and width of the lines. FCol and BCol determine the colour of the lines.

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

Note that if you specify filling, you do not have to define a **closed** polygon. The first and last points will automatically be joined for you if necessary.

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

The structure of the property values is best considered separately for single and multiple polylines or polygons.

Single Polyline or Polygon

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

LStyle and LWidth are both simple scalar numbers.

FStyle is either a single number specifying a standard fill pattern, or the name of a Bitmap object which is to be used as a "brush" to fill the polygon.

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

First make a Form :

```
'F' ⎕WC 'Form'
```

Draw a single line from (y=20, x=10) to (y=30, x=50)

```
'F.L1' ⎕WC 'Poly' ((20 30)(10 50))
```

or

```
L ← 2 2⍴20 10 30 50
'F.L1' ⎕WC 'Poly' L
```

Draw a horizontal line from (y=20, x=10) to (y=20, x=50). Note scalar extension of y-coordinate.

```
'F.L1' ⎕WC 'Poly' (20(10 50))
```

Draw an empty box in green :

```
Y ← 10 10 50 50 10
X ← 10 50 50 10 10
'F.L1' ⎕WC 'Poly' (Y X) (0 255 0)
```

Ditto, using a green/blue dashed line (LStyle 1) :

```
'F.L1' ⎕WC 'Poly' (Y X) (0 255 0)(0 0 255) 1
```

Draw a red filled rectangle with a black border 5 pixels wide :

```
'F.L1' ⎕WC 'Poly' (Y X) (0 0 0) ('LWidth' 5)
             ('FStyle' 0) ('FillCol' 255 0 0)
```

Multiple Polylines/Polygons

To draw a set of polylines or polygons with a single name, Points is a nested vector whose items are themselves 2-column matrices or 2-element nested vectors.

LStyle and LWidth may each be simple scalar values (applying to all the polylines) or simple vectors whose elements refer to each of the corresponding polylines in turn.

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

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

First make a Form :

```
'F' ⎕WC 'Form'
```

Draw two concentric triangles :

```
BY ← 10 10 50 10
BX ← 15 65 40 15

RY ← 15 15 40 15
RX ← 25 55 40 25

'F.L1' ⎕WC 'Poly' ((BY BX)(RY RX))
```

Or, using matrices :

```
BT ← BY,[1.5]BX
RT ← RY,[1.5]RX

'F.L1' ⎕WC 'Poly' (BT RT)
```

Ditto, but draw the first blue, the second red :

```
'F.L1' □WC 'Poly' (BT RT) ((0 0 255)(255 0 0))
```

Ditto, but make the lines 3 pixels wide :

```
'F.L1' □WC 'Poly' (BT RT) ((0 0 255)(255 0 0))
        ('LWidth' 3)
```

Ditto, but make the line widths 3 and 6 pixels respectively :

```
'F.L1' □WC 'Poly' (BT RT) ((0 0 255)(255 0 0))
        ('LWidth' 3 6)
```

Draw the first hollow, but fill the second in green :

```
'F.L1' □WC 'Poly' (BT RT) ('FStyle' ¯1 0)
        ('FillCol' (⊂0 255 0))
```

Popup Property

Applies to SysTrayItem, ToolButton

The Popup property specifies the name of a (popup) Menu object that is associated with a SysTrayItem or ToolButton.

Note that Popup is ignored unless Style is set to `'DropDown'`.

Posn   Property

Applies to   ActiveXControl, Animation, Button, Calendar, ColorButton, Combo,
                    ComboEx, CoolBand, CoolBar, DateTimePicker, Edit, Form, Grid,
                    Group, Label, List, ListView, Locator, MDIClient, Menu, MenuItem,
                    ProgressBar, PropertyPage, PropertySheet, RichEdit, Root, Scroll,
                    Separator, SM, Spinner, Splitter, Static, StatusBar, StatusField,
                    SubForm, TabBar, TabBtn, TabButton, TabControl, ToolBar,
                    ToolButton, ToolControl, TrackBar, TreeView, UpDown

For most objects to which it applies, Posn is a 2-element numeric vector specifying the
y-position and x-position respectively of the top-left corner of the object relative to its
parent. For a Form, Posn specifies its position on the screen. The units are defined by
the Coord property.

When specifying Posn for `⎕WC`, you can allow the y-position or x-position to assume a
default value by giving the corresponding element a value of `0`.

Using `⎕WS`, if you want to set the y-position, but not the x-position, or vice-versa, you
should specify `0` for the item you don't want to change.

For Menu, MenuItem and Separator objects, Posn is a single integer that specifies the
position at which the object is to be **inserted** in its parent. For example, to add a new
MenuItem between the third and fourth items in an existing Menu, you would specify its
Posn as 4. For these objects, the value of Posn returned by `⎕WG` is the current index of
the object within its parent.

PreCreate   Event 534

Applies to   ActiveXControl

If enabled, this event is reported when an instance of an ActiveXControl is created. The PreCreate event is generated at the point the *instance* is made.

An ActiveXControl also generates a Create event, which occurs *after* the PreCreate event at the point when the host application requires the instance to appear visually.

Note that at the time that PreCreate is generated, the ActiveXControl does not have a window.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to ¯1 or by returning 0 from a callback function.

The event message reported as the result of ⎕DQ, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	*'PreCreate'* or 534

Print   Method 100

Applies to   Printer

This method causes any spooled output to be printed.

The Print method is niladic.

If you attach a callback function to this event and have it return a value of 0, the printout will not be spooled.

Printer   Object

Purpose	To provide printer output.
Parents	ActiveXControl, CoolBand, Form, OLEServer, PropertyPage, PropertySheet, Root, TCPSocket
Children	Bitmap, Circle, Ellipse, Font, Icon, Image, Marker, Metafile, Poly, Rect, Text, Timer
Properties	Type, PName, DevCaps, Coord, Event, FontObj, FontList, YRange, XRange, Data, TextSize, EdgeStyle, Handle, Orientation, Copies, PrintRange, Collate, PaperSize, PaperSizes, PaperSource, PaperSources, ColorMode, Resolution, Resolutions, Duplex, Translate, Accelerator, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, Select
Methods	Abort, Detach, GetTextSize, NewPage, Print, RTFPrintSetup, Setup

The PName property is a character vector which specifies the name of an installed printer and the device to which it is attached. The name and device are separated by a comma (,). All valid values of PName can be obtained from the PrintList property of the Root object.

If not specified, the default value of PName is (⊃'.' ⎕WG '*PrintList*').

The DevCaps property reports the size of the printable area of the page in pixels (dots) and in millimetres. It also reports the number of colours available. This is 2 on a monochrome printer (black and white), although grey scales may be available.

The FontList property provides a list of fonts that are applicable and includes TrueType and printer fonts. This list is typically different from that obtained from the FontList property on the Root object which lists those fonts that apply to the screen.

The graphics objects listed above may be printed in much the same way as they may be displayed on a Form or Static. The differences are :

a) Once an object has been created, it will be printed, even if its name is subsequently expunged.

b) An object does not **replace** an existing one which has the same name.

c) The act of changing one or more properties of a named object causes the object to be printed a second time. For example, changing the Posn of an object will print it again at a different place.

In general it is recommended that you use unnamed objects for printing.

The Printer object supports five special methods :

Method	Number	Description
Print	100	Sends output to print spooler
Setup	101	Displays Printer Set-up dialog box
NewPage	102	Throws a new page
Abort	103	Aborts the print job
RTFPrintSetup	460	Displays Printer Set-up dialog box for RichEdit

Examples

Start a print job on the default printer

```
'PR1' ⎕WC 'Printer'
```

Write a centred heading at the top of the page using a proportional font

```
'PR1.' ⎕WC 'Text' 'Report Title' (0 50)
          ('HAlign' 1) ('FontObj' 'Roman' 64)
```

Draw a line across the page, 2 pixels wide

```
'PR1.' □WC 'Poly' (2(0 100)) ('LWidth' 2)
```

Print a character matrix. Note that a fixed width font is used.

```
REPORT ← 'I6' □FMT ?20 6ρ1000
'PR1.' □WC 'Text' REPORT (10 0)
        ('FontObj' 'Dyalog Std TT')
```

Throw a new page

```
PR1.NewPage
```

Spool output

```
□EX 'PR1'
```

PrintList   Property

Applies to   Root

This property provides a list of the printers that are installed on your computer system, i.e. those listed when you select "printers" from the Windows Control Panel. It is a "read-only" property of the Root object '.'.

PrintList is a vector of character vectors. Each item in PrintList contains the name of an installed printer followed by a comma (,) and then the name of the device to which it is attached. The first item in PrintList is the default system printer.

Example

```
     DISPLAY '.' □WG 'PrintList'
.→----------------------------------------.
| .→------------------------------------. |
| |Lexmark Optra S 2455 PS2,\\Pce20\biglex| |
| '------------------------------------' |
'∊----------------------------------------'
```

PrintRange   Property

Applies to   Printer

The PrintRange property specifies the range of pages to be printed.

PrintRange may be an empty character vector (the default), or `'All'`, either of which will cause all pages to be printed.

Alternatively, PrintRange may be a 3 or 4-element nested array whose items are:

> [1] `'Pages'`
>
> [2] Start page (integer)
>
> [3] End page (integer)
>
> [4] Maximum number of pages (integer)

In this case, printing starts at the page specified to be the Start page, and ends at the page specified by End page or after the Maximum number of pages has been reached, whichever is sooner.

ProgressBar   Object

Purpose	The ProgressBar object is used to indicate the progress of a lengthy operation.
Parents	ActiveXControl, CoolBand, Form, Group, PropertyPage, StatusBar, SubForm, ToolBar, ToolControl
Children	Bitmap, Circle, Cursor, Ellipse, Font, Icon, Marker, Poly, Rect, Text, Timer
Properties	Type, Posn, Size, Style, Coord, Active, Visible, Event, Thumb, Step, Wrap, Limits, Sizeable, Dragable, BCol, CursorObj, AutoConf, Data, Attach, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, Translate, Accelerator, AcceptFiles, KeepOnClose, ProgressStyle, Redraw, MethodList, ChildList, EventList, PropList
Events	Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, GotFocus, Help, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Select
Methods	Animate, Detach, GetFocus, GetTextSize, ProgressStep, ShowSIP

The ProgressBar object is a window that an application can use to indicate the progress of a lengthy operation.

It consists of a rectangle that is gradually filled, from left to right, with the system highlight colour as an operation progresses. The appearance of the rectangle is specified by the ProgressStyle property that may be *'Normal'* (the default) or *'Smooth'*.

The range of a ProgressBar is specified by the Limits property. This is a 2-element integer vector defining its minimum and maximum values. The position of the filled rectangle is specified by the Thumb property. You can update the ProgressBar by using $\square WS$ to set the value of the Thumb directly, or by generating a ProgressStep event. The latter causes the Thumb to be updated by the value of the Step property.

If you attempt to set the Thumb to a value greater than its maximum value (using either method) the behaviour depends upon the value of the Wrap property which is Boolean and has a default value of 1. If Wrap is 1, the value obtained when you set the Thumb property is given by the expression:

```
LIMITS[1]+(1+LIMITS[2]-LIMITS[1])|THUMB-LIMITS[1]
```

where $THUMB$ is the value to which you set the Thumb property and $LIMITS$ is the value of the Limits property. This causes the highlighted rectangle to begin filling again from the left.

There are two basic ways that you can use a ProgressBar. One is to fill the bar just once, doing so as evenly as possible and ensuring that the completion of the process coincides with the complete filling of the bar. This gives the user useful (and hopefully accurate) information as to the progress of the operation. However, some processes have an indeterminate duration (for example, an iterative one) and you may choose to use the ProgressBar simply to indicate that the operation is continuing.

Here is a code fragment illustrating how you can update a ProgressBar evenly. ($DO\_SOMETHING$ represents some processing that is performed N times in a loop):

```
'F.P'⎕WC'ProgressBar' ('Limits' 0 N)
I←0
Loop:→(N<I←I+1)/End
DO_SOMETHING I
'F.P' ⎕WS 'Thumb' I
→Loop
End:
```

and the same thing using the ProgressStep method:

```
'F.P'⎕WC'ProgressBar' ('Limits' 0 N) ('Step' 1)
I←0
Loop:→(N<I←I+1)/End
DO_SOMETHING I
F.P.ProgressStep
→Loop
End:
```

Alternatively, you can update a ProgressBar via a Timer.

ProgressStep Method 250

Applies to   ProgressBar

This method is used to increment the thumb in a ProgressBar object.

The ProgressStep is niladic.

The ProgressStep method causes the ProgressBar to attempt to increment its thumb by the value of its Step property, taking into account the setting of its Wrap property. If the values of the Thumb, Step and Limits properties are *THUMB*, *STEP* and *LIMITS* respectively, the new value of Thumb (and the corresponding position of the highlighted bar) is:

if Wrap is 0:

$$LIMITS[2]\lfloor THUMB+STEP$$

if Wrap is 1:

$$LIMITS[1]+(1+LIMITS[2]-LIMITS[1])\,|\,THUMB+STEP-LIMITS[1]$$

ProgressStyle   Property

Applies to   ProgressBar

The ProgressStyle property specifies the appearance of a ProgressBar control.

ProgressStyle is a character vector that may be '*Normal*' or '*Smooth*'. Its value is effective only when the object is created with *⎕WC*. Changing ProgressStyle with *⎕WS* has no effect on the appearance or behaviour of the ProgressBar.

If ProgressStyle is '*Normal*', the highlight in the centre of the ProgressBar is displayed as a broken bar. This is the default.

If ProgressStyle is '*Smooth*', the highlight in the centre of the ProgressBar is displayed as a solid block of colour.

The picture below illustrates the appearance of the different values of ProgressStyle.

PropertyPage   Object

Purpose   The PropertyPage object represents a single page in a PropertySheet.

Parents   PropertySheet

Children   Animation, Bitmap, BrowseBox, Button, Calendar, Circle, Clipboard, ColorButton, Combo, ComboEx, Cursor, DateTimePicker, Edit, Ellipse, FileBox, Font, Grid, Group, Icon, Image, ImageList, Label, List, ListView, Locator, Marker, Metafile, MsgBox, OCXClass, Poly, Printer, ProgressBar, Rect, RichEdit, Scroll, SM, Spinner, Splitter, Static, SubForm, TCPSocket, Text, Timer, TipField, TrackBar, TreeView, UpDown

Properties   Type, Caption, Posn, Size, Coord, Active, Event, HasHelp, FontObj, Data, EdgeStyle, Hint, HintObj, Tip, TipObj, Changed, Translate, AcceptFiles, KeepOnClose, MethodList, ChildList, EventList, PropList

Events   Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, FontCancel, FontOK, GotFocus, Help, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, PageActivate, PageApply, PageBack, PageCancel, PageChanged, PageDeactivate, PageFinish, PageHelp, PageNext, SetWizard

Methods   Animate, ChooseFont, Detach, GetFocus, GetTextSize, ShowSIP

The PropertyPage object represents a single page within a PropertySheet.

The Posn and Size properties are read-only properties determined by the parent PropertySheet and may not be changed using $\square WC$ or $\square WS$.

The HasHelp property is either 1 (the default) or 0. If the parent PropertySheet has a "Help" button (determined by its own HasHelp property) this property determines whether or not the Help button is active when the PropertyPage is the current page. If the HasHelp property of a PropertyPage is 0, the Help button on the parent PropertySheet will be temporarily disabled when that PropertyPage is displayed.

The PropertyPage object generates a PageActivate event when it becomes the current page and a PageDeactivate event when another page is selected. These events may not be disabled by a callback function.

If the user presses the Cancel button, the current PropertyPage generates a PageCancel event. This is followed by a Close event which is reported by the parent PropertySheet.

Other properties and behaviour depend upon the Style of the parent PropertySheet which may be `'Standard'` or `'Wizard'`

Standard Behaviour
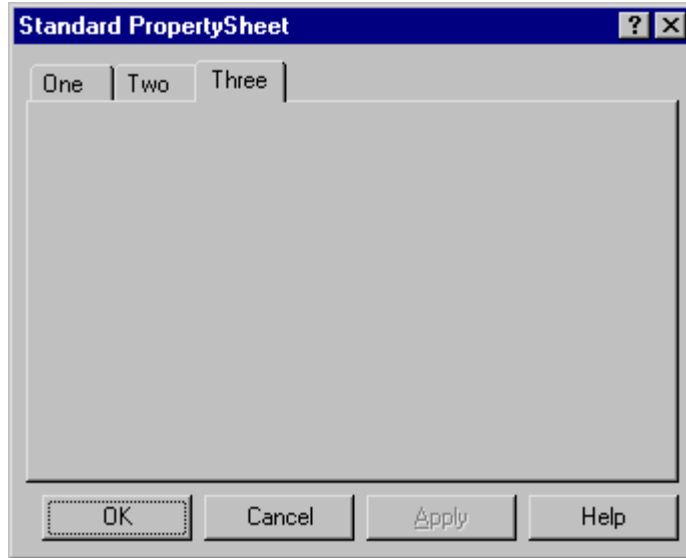
In a *Standard* PropertySheet, the Caption property of each PropertyPage specifies the text that is written in its tab.

PropertyPage objects owned by a Standard PropertySheet generate PageCancel, PageApply and PageHelp events. These events are all caused by the user pressing the corresponding button in the parent PropertySheet.

Conventionally, the Apply button is initially inactive. When the user changes an item on any of the PropertyPages, the Apply button immediately becomes active. When the user clicks the Apply button, the application responds (normally by changing the appropriate properties) and then the Apply button becomes inactive once again. This process is controlled as follows.

The Changed property is a Boolean value that determines whether or not a PropertyPage is marked as having been in any way altered. The Apply button is active if the value of the Changed property for *any* of the PropertyPages is 1, and is inactive otherwise

Initially, the value of the Changed property for all of the PropertyPages is 0 and the Apply button is therefore inactive. If the user alters a control on a PropertyPage, by, for example typing into an Edit object or changing the State of a Radio Button, the PropertyPage immediately generates a PageChanged event with the parameter 1. The default processing for this event is to set the Changed property of the PropertySheet (to 1). This in turn activates the "Apply" button. If you return 0 from a callback on the PageChanged event, the Changed property remains 0 and the Apply button remains inactive.

When the user clicks the Apply button, each of the PropertyPages whose Changed flag is currently set to 1 generates a PageApply event. The default processing for this event is to generate a PageChanged event with the parameter 0. This is turn resets the Changed property of the PropertyPage to 0. Once all of the Changed flags have been reset, the Apply button becomes inactive. If you return 0 from a callback on any of the PageChanged events, the Changed property for the corresponding PropertyPage remains 1 and the Apply button remains active.

You may control the value of the Changed property using $\Box WS$ or by calling PageChanged as a method. In all cases, the Apply button is active if the value of Changed on any PropertyPage is 1, and inactive otherwise.

Wizard Behaviour

If the PropertyPage is owned by a *Wizard* PropertySheet, its Caption property specifies the text that appears in the title bar of the PropertySheet window when the PropertyPage is the current page. Note that a Wizard PropertySheet ignores its own Caption property.

There are effectively 3 page changing buttons on a Wizard PropertySheet, named Back, Next and Finish. The Next and Finish buttons actually occupy the same position and are mutually exclusive. The captions on the buttons are language-dependent.

Conventionally, the buttons change according to which of the PropertyPages is currently displayed. If the first one is displayed, the Next button is active but the Back button is inactive. When a middle page is displayed, both the Next and Back buttons are active. When the last page is displayed, the caption on the Next button changes to Finish. However, in some applications, the Back button may be disabled to prevent the user returning to a previous page.

When the user clicks the Back or Next button, the PropertyPage generates a PageBack or PageNext event followed by a PageDeactivate event. The new PropertyPage then generates a PageActivate event. These are followed by a SetWizard event which is generated by the parent PropertySheet and actually controls the state of the buttons. When the user clicks the Finish button, the PropertyPage generates a PageFinish event alone. All of these events reported by the PropertyPage are reported for information only. Returning 0 from a callback function has no effect. You may however control the buttons using the SetWizard event.

PropertySheet Object

Purpose	The PropertySheet object represents a standard multi-page dialog box.
Parents	ActiveXControl, Form, OLEServer, Root, SubForm, TCPSocket
Children	Bitmap, BrowseBox, Circle, Clipboard, Cursor, Ellipse, FileBox, Font, Icon, Locator, Marker, Metafile, MsgBox, Poly, Printer, PropertyPage, Rect, Text, Timer, TipField
Properties	Type, Caption, Posn, Size, Style, Coord, Active, Visible, Event, HasApply, HasHelp, PageActive, FontObj, OnTop, Data, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, Translate, KeepOnClose, Redraw, MethodList, ChildList, EventList, PropList
Events	Close, Create, FontCancel, FontOK
Methods	CancelToClose, ChooseFont, Detach, GetFocus, SetFinishText, Wait

There are two different kinds of PropertySheet which you select using the Style property. This may only be set when the PropertySheet is created using ⎕*WC* and Style may not subsequently be changed using ⎕*WS*.

If Style is Standard (the default), the PropertySheet displays a set of pages (each represented by a PropertyPage) as a set of tabbed forms. The user selects the current page by clicking on the appropriate tab. This Style allows the user to select any page at any time and does not oblige the user to visit any but the first page you choose to display. This Style is useful for displaying groups of options or settings that the user may change.

If Style is Wizard, the PropertySheet displays its pages in succession starting with the first. The user steps from one to another using the Next and Back buttons and may be forced to visit all the pages in a prescribed order. This Style is useful for data entry or for asking the user to make a series of choices.

The Caption property specifies the text written in the window title bar, but only applies if the Style is Standard. The title bar text of a Wizard PropertySheet is specified by the Caption of the current PropertyPage. The FontObj and EdgeStyle properties have no effect on the appearance of the PropertySheet itself, but may be used to define the default appearance of its children.

The HasApply and HasHelp properties are Boolean and specify whether or not the PropertySheet has "Apply" and "Help" buttons respectively. These properties may only be set when the object is created using ⎕*WC*. They both have default values of 1.

PropList   Property

Applies to   ActiveXContainer, ActiveXControl, Animation, Bitmap, BrowseBox, Button, Calendar, Circle, Clipboard, ColorButton, Combo, ComboEx, CoolBand, CoolBar, Cursor, DateTimePicker, Edit, Ellipse, FileBox, Font, Form, Grid, Group, Icon, Image, ImageList, Label, List, ListView, Locator, Marker, MDIClient, Menu, MenuBar, MenuItem, Metafile, MsgBox, OCXClass, OLEClient, OLEServer, Poly, Printer, ProgressBar, PropertyPage, PropertySheet, Rect, RichEdit, Root, Scroll, Separator, SM, Spinner, Splitter, Static, StatusBar, StatusField, SubForm, SysTrayItem, TabBar, TabBtn, TabButton, TabControl, TCPSocket, Text, Timer, TipField, ToolBar, ToolButton, ToolControl, TrackBar, TreeView, UpDown

This is a "read-only" property that supplies a list of all other properties which are applicable to the object in question. The list is returned as a vector of character vectors in the order in which the corresponding properties are expected by *⎕WC* and *⎕WS*.

Protected   Event 470

Applies to   RichEdit

If enabled, this event is reported when the user attempts to alter protected text in a RichEdit. See CharFormat property.

The event message reported as the result of *⎕DQ*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	*'Protected'* or 470

QueueEvents   Property

Applies to   OCXClass, OLEClient

The QueueEvents property specifies whether or not incoming events generated by a COM object are queued.

QueueEvents is a single number with the value 1 (queue events) or 0 (process events immediately). The default value is 1.

If QueueEvents is 1, the result (if any) of your callback function is **not** passed back to the COM object but is discarded. Thus you cannot, for example, inhibit or modify the default processing of the event by the COM object.

If QueueEvents is 0, the following applies.

The callback function attached to the event is executed **immediately**, even if there are other APL events before it in the internal event queue. This immediate execution means that your callback can fire during the execution of any other function, including a callback function on an APL event. You must therefore take care that the callback makes no references to objects that may be shadowed.

The result of your callback function is then passed back to the COM object. In this situation, it is essential that the callback is not interrupted by other events from the same, or another instance, of an COM object.

To prevent APL itself from yielding to Windows, the Yield property is temporarily and automatically set to 0 while the callback is run. For the same reason, the tracing of a callback function, that is run immediately in this way, is disabled.

However, you must yourself also ensure that your own code does not yield. This means that you may not perform any operation in your callback that would yield to Windows; these include:

- $\Box DL$
- certain uses of $\Box NA$
- external function calls to Auxiliary Processors

If your callback does yield to Windows, thereby allowing another COM object event to arrive, this second event and any subsequent events that arrive during the execution of the callback are queued and will be processed later. These events may therefore not be modified by their callback functions.

Radius   Property

Applies to   Circle, Rect

For a Circle object, this property is a single number that specifies the radius of the circle/arc or a numeric vector that specifies the radii of a set of circles/arcs.

For a Rect object, Radius is a 2-element vector that specifies the curvature of the corners of the rectangle or set of rectangles to be drawn. The curvature is defined in terms of the vertical and horizontal axes of an ellipse. The first element of Radius defines the axis vertically, the second horizontally. If more than one rectangle is involved, either or both of the elements of Radius may be vectors. The default value is (0,0) which gives square corners.

Range   Property

Applies to   Form, Scroll, SubForm

This property determines the maximum value of the thumb in a scrollbar (the minimum value is always 1). This may be any positive integer value that is greater than 1.

For a Scroll object Range is a single number. For a Form or SubForm object, Range is a 2-element vector which specifies the maxima for the Form's vertical and horizontal scrollbars respectively.

ReadOnly   Property

Applies to   Button, Edit, Spinner

This property specifies whether or not the user may alter the text in an Edit or Spinner object or the state of a radio button or checkbox. The default value of ReadOnly is 0 which allows the user to alter text.

For an Edit or Spinner, if you set ReadOnly to 1, a cursor is displayed in the object, the user may navigate around the text in the usual manner with the mouse and/or the keyboard and select text and copy it to the clipboard. However, all input that would otherwise change the data is ignored.

For a Button object with Style *'Radio'* or *'Check'*, setting ReadOnly to 1 prevents the user from changing the state of the Button, although mouse and other events will still be reported.

RealSize   Property

Applies to   Metafile

There are several distinct types of Windows metafiles. A *placeable* metafile is one that carries with it its *suggested size*. Certain programs (such as Word for Windows) only support placeable metafiles.

The RealSize property specifies the suggested size of a Metafile in units of 0.01mm. Thus to make a placeable Metafile with a suggested size of 20 x 10 cm, you would set RealSize to (20000 10000).

The RealSize property is not used or required by Dyalog APL/W and is provided only to enable you to make and save a new metafile that is placeable. If you create a Metafile object from a file, the value of RealSize will be obtained from the value recorded in the file (if it is placeable). Otherwise, RealSize will be (0 0). If so, you must set RealSize to make it placeable. Each element of RealSize must be an integer in the range 0-144745.

Rect   Object

Purpose	A graphical object used to draw boxes
Parents	ActiveXControl, Animation, Bitmap, Button, Combo, ComboEx, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, Metafile, Printer, ProgressBar, PropertyPage, PropertySheet, RichEdit, Scroll, Spinner, Static, StatusBar, SubForm, TabBar, TipField, ToolBar, TrackBar, TreeView, UpDown
Children	Timer
Properties	Type, Points, Size, Radius, FCol, BCol, LStyle, LWidth, FStyle, FillCol, Coord, Visible, Event, Dragable, OnTop, CursorObj, AutoConf, Data, EdgeStyle, Accelerator, KeepOnClose, DrawMode, MethodList, ChildList, EventList, PropList
Events	Close, Create, DragDrop, Help, MouseDblClick, MouseDown, MouseMove, MouseUp, Select
Methods	Detach

The Points property specifies one or more sets of co-ordinates which define the position(s) of one or more rectangles. The position of a rectangle is defined to be the position of the corner that is **nearest** to the origin of its parent. The default is therefore its top-left corner. The Size property specifies the height and width of each rectangle, measuring away from the origin.

The Radius property specifies the curvature of the corners of the rectangle.

LStyle and LWidth define the style and width of the lines used to draw the boundaries of the rectangle(s). FCol and BCol determine the colour of the lines.

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

The EdgeStyle property may specify a 3-dimensional effect. If so, the boundary line around the rectangle is replaced by a border designed to achieve the desired effect.

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

Single Rectangle

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

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

Radius is a 2-element vector which specifies the major (y-axis) and minor (x-axis) radii of an ellipse used to draw the corners of the rectangle. Its default value is (0 0) which yields right-angled corners.

LStyle and LWidth are both simple scalar numbers.

FStyle is either a single number specifying a standard fill pattern, or the name of a Bitmap object which is to be used as a "brush" to fill the rectangle.

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

First make a Form :

```
'F' ⎕WC 'Form'
```

Draw a single rectangle at (y=10, x=5) with height=30, width=50 :

```
'F.R1' ⎕WC 'Rect' (10 5)(30 50)
```

Ditto with rounded corners (radii 10) :

```
'F.R1' ⎕WC 'Rect' (10 5)(30 50)(10 10)
```

Ditto, but use a red line :

```
'F.R1' ⎕WC 'Rect' (10 5)(30 50)(10 10)
             (255 0 0)
```

Ditto, but fill in green

```
'F.R1' ⎕WC 'Rect' (10 5)(30 50)(10 10)
             (255 0 0) ('FStyle' 0)(0 255 0)
```

Multiple Rectangles

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

Likewise, Size may be a simple 2-element vector (applying to all the rectangles), **or** a 2-column matrix whose first column specifies their heights and whose second column specifies their widths, **or** a 2-element nested vector whose first element specifies their height(s) and whose second element specifies their width(s).

Radius may be a simple 2-element vector (applying to all the rectangles), **or** a 2-column matrix whose first column specifies major radii and whose second column specifies minor radii, **or** a 2-element nested vector whose first element specifies major radii and whose second element specifies minor radii.

LStyle and LWidth may each be simple scalar values (applying to all the rectangles) or simple vectors whose elements refer to each of the corresponding rectangles in turn.

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

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

First make a Form :

```
'F' ⎕WC 'Form'
```

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

```
'F.R1' ⎕WC 'Rect' ((5 5)(10 60)) (40 10)
```

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

```
'F.R1' ⎕WC 'Rect' (5(10 60)) (40 10)
```

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

```
'F.R1' ⎕WC 'Rect' (5(10 60)) ((40 20)(30 10))
```

Draw two rectangles at (y=5, x=10) and (y=5, x=60) each of (height=40, width=10) and with rounded corners of radii (10,10) :

```
'F.R1' ⎕WC 'Rect' (5(10 60)) (40 10) (10 10)
```

Ditto, using a green line for both :

```
'F.R1' ⎕WC 'Rect' (5(10 60)) (40 10) (10 10)
            (⊂0 255 0)
```

Ditto, but using red and blue lines respectively :

```
'F.R1' ⎕WC 'Rect' (5(10 60)) (40 10) (10 10)
            ((255 0 0)(0 0 255))
```

Redraw   Property

Applies to   ActiveXControl, Button, Calendar, ColorButton, Combo, ComboEx, CoolBar, DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, ProgressBar, PropertySheet, RichEdit, Scroll, SM, Spinner, Static, StatusBar, SubForm, TabBar, ToolBar, ToolControl, TrackBar, TreeView, UpDown

The Redraw property specifies whether or not APL automatically redraws an object when it is exposed or when any of its properties change in a way that would affect its appearance.

The value reported by the Redraw property is a Boolean value; 1 means that APL automatically redraws the object when necessary (the default); 0 means that APL does not redraw the object.

Setting Redraw to 0 or 1 affects only whether or not APL will redraw the object from then on.

In addition to the values 0 and 1, you may set Redraw to 2. This has the same effect as setting it to 1, but object is also redrawn immediately.

RemoteAddr Property

Applies to TCPSocket

The RemoteAddr property is a character vector that specifies the IP address of the remote computer.

RemoteAddr may only be specified by a client TCPSocket that is intended to make a connection with a server. Furthermore, it must be specified in the `⎕WC` statement that creates the TCPSocket object and it may not subsequently be changed using `⎕WS`.

Note that you may use *either* RemoteAddr *or* RemoteAddrName to identify the remote computer. If you know its IP address, it is normally quicker to specify RemoteAddr. If you specify both properties, the value of RemoteAddrName will be ignored.

For a server TCPSocket, RemoteAddr is determined by the IP address of the connecting process and is a read-only property and is available reliably only after connection.

RemoteAddrName Property

Applies to TCPSocket

The RemoteAddrName property is a character vector that specifies the host name of the remote computer to which you wish to make a connection.

RemoteAddrName may only be specified by a client TCPSocket that is intended to make a connection with a server. Furthermore, it must be specified in the `⎕WC` statement that creates the TCPSocket object and it may not subsequently be changed using `⎕WS`.

When the specified host name has been resolved to an IP address, the TCPSocket will generate a TCPGotAddr event and update the value of RemoteAddr accordingly.

Note that you may use *either* RemoteAddr *or* RemoteAddrName to identify the remote computer. If you know its IP address, it is normally quicker to specify RemoteAddr. If you specify both properties, the value of RemoteAddrName will be ignored.

For a server TCPSocket, you may not specify RemoteAddrName and `⎕WG` returns an empty character vector.

RemotePort   Property

Applies to   TCPSocket

The RemotePort property is a scalar integer in the range 1-65535 that identifies the port number associated with a service on a remote computer.

RemotePort may only be specified by a client TCPSocket that is intended to make a connection with a server. Furthermore, it must be specified in the ⎕*WC* statement that creates the TCPSocket object and it may not subsequently be changed using ⎕*WS*.

Note that you may use *either* RemotePort *or* RemotePortName to identify the remote service. If you know the port number, it is normally quicker to specify RemotePort. However unless it is a *well known port number*, the use of a port name is generally more flexible. If you specify both properties, the value of RemotePortName will be ignored.

For a server TCPSocket, RemotePort is determined by the port number of the connecting process and is a read-only property.

RemotePortName   Property

Applies to   TCPSocket

The RemotePortName property is a character vector that specifies the port name of the remote service to which you wish to make a connection.

RemotePortName may only be specified by a client TCPSocket that is intended to make a connection with a server. Furthermore, it must be specified in the ⎕*WC* statement that creates the TCPSocket object and it may not subsequently be changed using ⎕*WS*.

When the specified port name has been resolved to a port number, the TCPSocket will generate a TCPGotPort event and update the value of RemotePort accordingly.

Note that you may use *either* RemotePort *or* RemotePortName to identify the remote service. If you know the port number, it is normally quicker to specify RemotePort. However unless it is a *well known port number*, the use of a port name is generally more flexible. If you specify both properties, the value of RemotePortName will be ignored.

For a server TCPSocket, you may not specify RemotePortName and ⎕*WG* returns an empty character vector.

ReportInfo   Property

Applies to   ListView

The ReportInfo property is a matrix that is displayed alongside each item in a ListView object when its View property is *'Report'*. Each element of the matrix may be a character vector or a number.

The information is displayed in a grid format, the first column of which contains the item labels and their icons. Subsequent columns of the grid are defined by the corresponding columns of ReportInfo. The alignment of the columns is specified by the ColTitleAlign property.

ResizeCols   Property

Applies to   Grid

This property determines whether or not the user may resize columns in the Grid. It is a Boolean scalar or vector with one element per column. A value of 1 indicates that the corresponding column is resizable by the user. A value of 0 means that the corresponding column may not be resized by the user.

If a column is resizable, the cursor changes to a double headed arrow when the mouse pointer is placed over the right-hand border of the column title. The user may resize the column by dragging this border. The user may also resize a column by double-clicking over its right-hand border. This causes the column to be resized to fit the data and the width of the column is automatically adjusted to display the widest value in any of its cells. Either operation generates a SetColSize event.

Note that the user may cause the column to disappear altogether by dragging it to a zero width. Once this has been done, this column may only be restored if the column to its left is itself not resizable.

ResizeColTitles   Property

Applies to   Grid

This property determines whether or not the user may alter the height of the column titles in the Grid. It is either 1, which indicates that the height of the column titles is adjustable by the user, or 0 which means that it is not.

If the height of the column titles is adjustable, the cursor changes to a double headed arrow when the mouse pointer is placed over the top border of the first row title The user may resize the column titles by dragging this border. The user may also resize the column titles by double-clicking over this border. This causes the column titles to be resized to fit the data and the height of the column titles is automatically adjusted to display the tallest heading in any of its columns. Either operation generates a SetRowSize event. The value of the row number reported by the event is ¯1.

Note that the user may cause the column titles to disappear altogether by dragging them to a zero height. Once this has been done, the row titles cannot be restored.

ResizeRows   Property

Applies to   Grid

This property determines whether or not the user may resize rows in the Grid. It is a Boolean scalar or vector with one element per column. A value of 1 indicates that the corresponding row is resizable by the user. A value of 0 means that the corresponding row may not be resized by the user.

If a row is resizable, the cursor changes to a double headed arrow when the mouse pointer is placed over the lower border of the row title. The user may change the height of the row by dragging this border up and down. The user may also resize a row by double-clicking over its bottom border. This causes the row to be resized to fit the data and the height of the row is automatically adjusted to display the tallest value in any of its cells. Either operation generates a SetRowSize event.

Note that the user may cause the row to disappear altogether by dragging it to a zero height. Once this has been done, this row may only be restored if the row above it is itself not resizable.

ResizeRowTitles   Property

Applies to   Grid

This property determines whether or not the user may alter the width of the row titles in the Grid. It is either 1, which indicates that the width of the row titles is adjustable by the user, or 0 which means that it is not.

If the width of the row titles is adjustable, the cursor changes to a double headed arrow when the mouse pointer is placed over the left-hand border of the first column title. The user may resize the row titles by dragging this border. The user may also resize the row titles by double-clicking over this border. This causes the row titles to be resized to fit the data and the width of the row titles is automatically adjusted to display the longest string in any of its rows. Either operation generates a SetColSize event. The value of the column number reported by the event is ¯1.

Note that the user may cause the row titles to disappear altogether by dragging them to a zero width. Once this has been done, the row titles cannot be restored.

Resolution   Property

Applies to   Printer

The Resolution property determines the print resolution.

You may set Resolution to `'Draft'`, `'Low'`, `'Medium'` or `'High'`.

Alternatively, you can set Resolution to a 2-element integer vector that specifies the desired number of dots per inch in the x (horizontal) and y (vertical) direction respectively.

The initial value reported by Resolution may be reported in either form (character vector or 2-element numeric vector) according to the current printer settings.

Resolutions   Property

Applies to   Printer

The Resolutions property is a read-only property that reports the available printer resolutions.

Resolutions is a vector of 2-element integer vectors each of which specifies the number of dots per inch in the x (horizontal) and y (vertical) directions respectively.

Retracting   Event 304

Applies to   TreeView

If enabled, this event is reported by a TreeView object just before it is about to retract to hide the children of the current item. This occurs when the user double-clicks the item label or clicks in the button or on the tree line to the left of the item label, when the item is in its expanded state. The default processing for the event is to retract the tree at the corresponding point.

You may disable the retract operation by setting the action code for the event to ¯1. You may also prevent the retraction from occurring by returning 0 from a callback function. You may retract a TreeView dynamically under program control by calling Retracting as a method.

The event message reported as the result of □*DQ*, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	'*Retracting*' or 304
[3] Item number:	Integer. The index of the item.

RichEdit   Object

Purpose	The RichEdit object is a multi-line text editor that provides a wide range of word-processing capabilities.
Parents	ActiveXControl, CoolBand, Form, Group, PropertyPage, SubForm, ToolBar, ToolControl
Children	Bitmap, Circle, Cursor, Ellipse, Font, Icon, Marker, Poly, Rect, Text, Timer
Properties	Type, Text, Posn, Size, File, Coord, Border, Active, Visible, Event, VScroll, HScroll, SelText, Sizeable, Dragable, FontObj, FCol, BCol, CursorObj, AutoConf, Data, Attach, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, Changed, RTFText, Translate, Accelerator, CharFormat, WordFormat, ParaFormat, PageWidth, AcceptFiles, WantsReturn, KeepOnClose, Redraw, MethodList, ChildList, EventList, PropList
Events	Change, Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, FontCancel, FontOK, GotFocus, Help, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Protected, Select
Methods	Animate, ChooseFont, Detach, FileRead, FileWrite, GetFocus, GetTextSize, RTFPrint, RTFPrintSetup, ShowSIP

A RichEdit object is a window in which the user can enter and edit text. The text can be assigned character and paragraph formatting.

The RichEdit object provides a programming interface for formatting text. However, your application must implement any user interface components necessary to make formatting operations available to the user. For example, your program can set the colour and font of a particular block of text, but the RichEdit itself provides no facilities for the user to do this directly. It is up to you to provide these.

The File property specifies the name of a file associated with the object. Data in the file is assumed to be in rich text format, and the default extension for the file is .RTF. You can read the file into the object by calling FileRead and you can write the contents to the file by calling FileWrite. You can also print the contents of the object by calling RTFPrint.

The Text property may be used to set or retrieve the text of the RichEdit, but ignores formatting information.

The RTFText property may be used to set or retrieve the contents of the RichEdit, including text and formatting.

The PageWidth property defines the width of the text within the object. Text entered into the object is automatically wrapped according to PageWidth. This property also defines the width when the text is printed.

You can set the default character format or the format of a particular block of text using the CharFormat property. If there is no selection, setting CharFormat defines the default character format that applies at the current insertion position and establishes the appearance of all of the text (font, colour, size etc.) that the user subsequently enters here. If there *is* a selection, setting CharFormat sets the character format for the selected block of text.

The WordFormat property is similar to CharFormat except that is sets the format for the selected word(s) or, if there is no selection, for the word containing the insertion point.

The ParaFormat property defines the paragraph formatting which includes alignment, indentation and the location of tab stops. When you set ParaFormat with $\Box WS$, the formatting is applied to the current selection. If there is no selection, it defines the default paragraph formatting at the insertion point.

All of the dimensions used for text and paragraph formatting are specified in Twips. You can convert from pixels to Twips and vice versa using the DevCaps property of either Root or the Printer object as appropriate.

The behaviour of the Enter key is defined by the WantsReturn property. If WantsReturn is 1 (the default), the Enter key inputs a new line into the RichEdit object. If WantsReturn is 0 the Enter key is ignored by the RichEdit object and may instead generate a Select event on a Button. In this case the user must press Ctrl+Enter to input a new line.

The user may copy and paste information (in RTF format) between a RichEdit object and the Windows clipboard. The Clipboard object also has an RTFText property that supports RTF format

If the user attempts to alter text that is protected (see CharFormat) the RichEdit object reports a Protected event.

You may print the contents of a RichEdit object using the RTFPrint method. You may display a print set-up dialog box using the RTFPrintSetup method.

Root   Object

Purpose	This is an invisible "system" object that acts as the parent of all other objects.
Parents	(None)
Children	Bitmap, BrowseBox, Clipboard, Cursor, FileBox, Font, Form, Icon, ImageList, Locator, Menu, Metafile, MsgBox, NetClient, NetType, OCXClass, OLEClient, OLEServer, Printer, PropertySheet, SysTrayItem, TCPSocket, Timer, TipField
Properties	Type, Caption, Posn, Size, DevCaps, Coord, Event, FontObj, FontList, PrintList, IconObj, CursorObj, YRange, XRange, Data, TextSize, Yield, EdgeStyle, HintObj, TipObj, Translate, UpperCase, APLVersion, KeepOnClose, OLEControls, OLEServers, LastError, MethodList, ChildList, EventList, PropList
Events	ActivateApp, DDE, DisplayChange, ExitApp, ExitWindows, FontCancel, FontOK, Idle, SysColorChange, WinIniChange
Methods	ChooseFont, DateToIDN, DeleteTypeLib, Flush, GetBuildID, GetCommandLine, GetCommandLineArgs, GetEnvironment, GetFocus, GetTextSize, GreetBitmap, IDNToDate, ListTypeLibs, NameFromHandle, ShowSIP, TCPGetHostID, Wait

There is a single Root object called `' . '` which is always present. It cannot be created using `⎕WC` nor can it be destroyed.

The Caption and IconObj properties of `' . '` are used to identify a Dyalog APL/W application as distinct from the APL Session. The Caption property specifies the application name that is displayed when you cycle through running applications using Alt+Tab and by the Windows Task List. The IconObj property specifies the name of an Icon object that is displayed alongside the application name in the box displayed by Alt+Tab. For these to take effect, your application must have at least one visible and active Form.

For the Root object, the value of Posn is (0,0). The value of Size is either (100,100) if Coord is `'Prop'`, or the size of the screen in pixels if Coord is `'Pixel'`. XRange and YRange both have the value (0,100). The DevCaps property reports the physical size of the screen in terms of both pixels and millimetres. It also reports the number of colours available. The FontList property provides a list of all the character fonts that are available. The PrintList property provides a list of all the installed printers. These properties are *read-only* and may not be changed.

As the default value of Coord is `'Inherit'` for all other objects, the value of Coord for `'.'` defines the default co-ordinate system. It may be either `'Prop'` (the default) or `'Pixel'`. `'Inherit'` and `'User'` are not allowed.

The CursorObj property is used to define a cursor for the application as a whole. Its default value is an empty character vector. If it is set to any value other than `''` or 0, the selected cursor overrides the CursorObj values for all other objects. If you want to indicate that the application is "busy", you can therefore set the CursorObj property on `'.'` to an hourglass for the duration of the operation, e.g.

         `'.' ⎕WS 'CursorObj' 1  ⍝ Set cursor to an hourglass`

   [lengthy process]

         `'.' ⎕WS 'CursorObj' 0  ⍝ Reset cursor`

The Yield property specifies how frequently APL yields to Windows during the execution of code. Its default value is 200 milliseconds.

The EdgeStyle property is used to determine whether or not objects may have 3-dimensional effects. Setting EdgeStyle to `'None'` disables 3-dimensional effects on all Forms and controls. Setting EdgeStyle to any other value enables 3-dimensional effects for these objects.

The ExitApp and ExitWindows events can be used to prevent the user closing your application from the Windows Task List or by terminating Windows.

The expression `⎕EX '.'` deletes all objects owned by the current thread **except** the Root object itself. In addition, if this expression is executed by thread 0, it resets all the properties of `'.'` to their default values.

Rotate   Property

Applies to   Font

This property specifies the angle of rotation of the font measured in radians (0 → ○2) from the x-axis in a counter-clockwise direction. Note that only TrueType fonts can be rotated. Rotated fonts are supported **only** for use with the Text object.

RowChange   Method 158

Applies to   Grid

This method is used to change the data in a row of a Grid object.

The argument to RowChange is a 2-element array as follows.

      [1] Row number: integer

      [2] Row data: array

Row data must be a scalar or a vector whose length is equal to the number of columns in the Grid. Its elements may be scalar numbers, character vectors or matrices.

RowLineTypes   Property

Applies to   Grid

This property specifies the appearance of the horizontal grid lines in a Grid object.

RowLineTypes is an integer vector, whose length is normally equal to the number of rows in the Grid. Each element in RowLineTypes specifies an index into the GridLineFCol and GridLineWidth properties, thus selecting the colour and width of the horizontal grid lines.

For example, if RowLineTypes[1] is 3, the first horizontal grid line in the Grid is displayed using the colour specified by the 3rd element of GridLineFCol, and the width specified by the 3rd element of GridLineWidth. Note that RowLineTypes is not $\square IO$ dependant, and the value 0 is treated the same as the value 1; both selecting the *first* colour and line width specified by GridLineFCol and GridLineWidth respectively.

The default value of RowLineTypes is an empty numeric vector (θ). If so, all horizontal grid lines are drawn using the first element of GridLineFCol and GridLineWidth.

A horizontal grid line is drawn along the bottom edge of its associated row. One pixel is drawn *inside* the row of cells; additional pixels (if any) are drawn *between* that row of cells and the next one below.

Rows Property

Applies to Combo, ComboEx

For Combo objects with Style `'Drop'` or `'DropEdit'` this property determines the number of rows displayed in the drop-down listbox when it is displayed. Note that the height of the edit field of a Combo of this type is dependent only upon the size of the font in use, and cannot otherwise be changed.

Rows is a "read-only" property for a Combo with Style `'Simple'` and an attempt to set it in a Combo of this type with `⎕WC` or `⎕WS` will generate a *NONCE ERROR*. Instead, the overall height of a Simple Combo is determined by the first element of the Size property.

RowTitleAlign Property

Applies to Grid

The RowTitleAlign property specifies the alignment of row titles in a Grid. It is either a simple character vector, or a vector of character vectors with one element per row.

An element of RowTitleAlign may be: `'Top'`,`'Bottom'`,`'Left'`,`'Right'`, `'Centre'`,`'TopLeft'`,`'TopRight'`,`'BottomLeft'`, or `'BottomRight'`. Note that both spellings `'Centre'` and `'Center'` are accepted.

RowTitleBCol Property

Applies to Grid

The RowTitleBCol property specifies the background colour of the row titles in a Grid object

RowTitleBCol may be a scalar that specifies a single background colour to be used for all of the row titles, or a vector that specifies the background colour of each of the row titles individually. An element of RowTitleBCol may be an enclosed 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or it may be a scalar that defines a standard Windows colour element (see BCol for details). Its default value is 0 which obtains the colour defined for Button Face.

RowTitleDepth   Property

Applies to   Grid

RowTitleDepth specifies the structure of a set of hierarchical row titles. It is an integer vector with the same length as the RowTitles property. A value of 0 indicates that the corresponding element of RowTitles is a top-level title. A value of 1 indicates that the corresponding title is a sub-title of the most recent title whose RowTitleDepth is 0; a value of 2 indicates that the corresponding title is a sub-title of the most recent title whose RowTitleDepth is 1, and so forth. For example:

```
'F'⎕WC'Form'('Coord' 'Pixel')('Size' 318 310)
F.Caption ← 'Hierarchical Column Titles'
'F.G'⎕WC'Grid'(?12 4ρ100)(0 0)(318 310)
F.G.TitleWidth ← 150
F.G.TitleHeight ← 0
F.G.CellWidths ← 40

Q1←'Q1' 'Jan' 'Feb' 'Mar'
Q2←'Q2' 'Apr' 'May' 'Jun'
Q3←'Q3' 'Jul' 'Aug' 'Sep'
Q4←'Q4' 'Oct' 'Nov' 'Dec'
RT←(⊂'1995'),Q1,Q2,Q3,Q4
RD←0,16ρ1 2 2 2

F.G.RowTitles F.G. RowTitleDepth ← RT RD
F.G.RowTitleAlign ← 'Centre'
```

RowTitleFCol   Property

Applies to   Grid

The RowTitleFCol property specifies the colour of the row titles in a Grid object

RowTitleFCol may be a scalar that specifies a single colour to be used for all of the row titles, or a vector that specifies the colour of each of the row titles individually. An element of RowTitleFCol may be an enclosed 3-element vector of integer values in the range 0-255 which refer to the red, green and blue components of the colour respectively, or it may be a scalar that defines a standard Windows colour element (see BCol for details). Its default value is 0 which obtains the colour defined for Button text.

RowTitles   Property

Applies to   Grid

This property specifies the headings that are displayed to the left of the rows in a Grid object. If specified, it must be a vector of character vectors or matrices whose length is the same as the number of rows implied by the Values property. The default value of RowTitles is an empty character vector. In this case, the system displays the row numbers.

To disable the display of row titles in a Grid, you should set the TitleWidth property to 0.

RTFPrint   Method 461

Applies to   RichEdit

This method is used to print the contents (RTFText) of a RichEdit object.

The argument to RTFPrint is 0, or a 1 to 4-element array as follows:

[1] Printer name:	Optional - character vector (see below)
[2] Print range:	Optional - (see below)
[3] Number of copies:	Optional - Integer.
[4] Collate:	Optional - 0 or 1

Printer name may be the name of an existing Printer object, or the (Windows) name of an installed printer. If you use the latter, the document will be spooled immediately. An empty vector implies the default printer.

Print range may be a simple character vector containing *'All'*, *'Pages'*, or *'Selection'*. Alternatively, it may be a 3 or 4-element nested vector containing:

[1] *'All'*, *'Pages'*, or *'Selection'*
[2] Start page (integer)
[3] End page (integer)
[4] Maximum pages (ignored)

RTFPrintSetup Method 460

Applies to Printer, RichEdit

This method is used to display a print set-up dialog box. The dialog box allows the user to select a particular printer, the pages to be printed and other information. The user's choices are returned in the result.

The argument to RTFPrintSetup is ⍬, or a 1 to 3-element array as follows:

      [1] Print range: Optional - (see below)

      [2] Number of copies: Optional - Integer.

      [3] Collate: Optional - 0 or 1

Print range may be a simple character vector containing *'All'*, *'Pages'*, or *'Selection'*. Alternatively, it may be a 3 or 4-element nested vector containing:
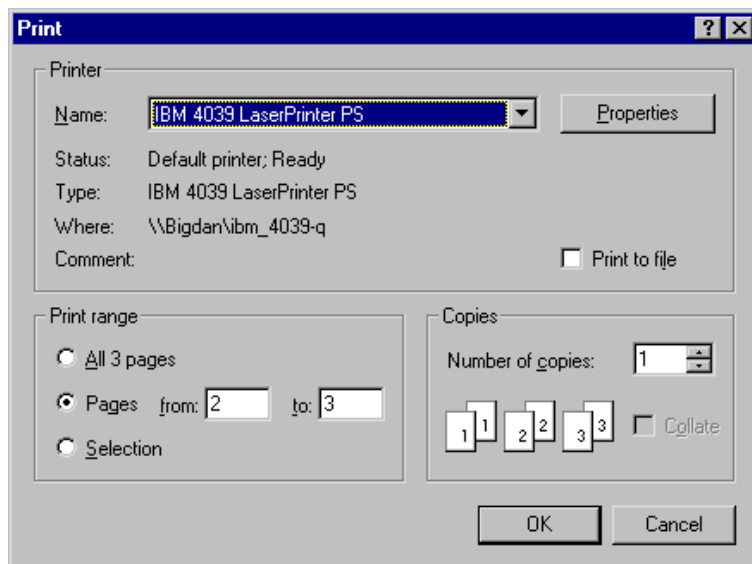
[1] *'All'*, *'Pages'*, or *'Selection'*
[2] Start page (integer)
[3] End page (integer)
[4] Maximum pages

Maximum pages (4th element of Print range) may be an integer number, or the name of a reference object. The latter allows the system to calculate the total number of pages required. If the object to which the RTFPrintSetup event is sent is a RichEdit, this is the name of a printer object. If the object to which the RTFPrintSetup event is sent is a Printer, this is the name of a RichEdit object. Both are required because the number of pages of a printed document is dependent upon both the content of the document and the characteristics of the device upon which it will be printed.

If the user presses OK, the result is a 4-element vector containing the user's choices as follows, otherwise the result is empty.

[1] Printer name:	character vector
[2] Print range:	(see above)
[3] Number of copies:	Integer.
[4] Collate:	0 or 1

Example

```
      F.T.RTFPrintSetup ('All' 1 1 'PR')
 IBM 4039 LaserPrinter PS   Pages  2 3 3  1 0
```

RTFText   Property

Applies to   Clipboard, RichEdit

The RTFText property is used to set or retrieve the contents of a Clipboard or a RichEdit object in rich text format (RTF). It is always a character vector.

RunMode   Property

Applies to   OLEServer

This property specifies the way in which an OLEServer object serves multiple clients. RunMode is a character vector and may be `'MultiUse'` (the default), `'SingleUse'` or `'RunningObject'`.

If RunMode is `'MultiUse'`, OLE will load a single copy of Dyalog APL and the appropriate workspace into memory. All OLE client processes will communicate with the same Dyalog APL session.

Note that in this case, each OLE client is actually connected to a separate *instance* of the corresponding APL namespace. That is to say, each client will appear to have its own private copy of the namespace. However, the individual functions and variables in the namespace are not physically copied until they are changed. This means that, in general, OLE clients will share APL functions but have private copies of the namespace variables. However, please remember that global objects in the workspace or in other namespaces are not *instanced* and will effectively be shared by all clients although they are not directly accessible to them.

If RunMode is `'SingleUse'`, OLE will load a separate copy of Dyalog APL and a separate copy of the appropriate workspace into memory for each OLE client. Each OLE client operates directly on the namespace associated with the object and not an *instance* of it.

If RunMode is `'RunningObject'`, OLE will load a single copy of Dyalog APL and the appropriate workspace into memory. All OLE client processes will communicate with the same Dyalog APL session and indeed with the same namespace. The namespace is not *instanced* and all objects, including exported variables, are shared by all clients.

Scroll   Object

Purpose   Provides a vertical or horizontal scrollbar

Parents   ActiveXControl, CoolBand, Form, Group, PropertyPage, SubForm, ToolBar, ToolControl

Children   Circle, Cursor, Ellipse, Font, Marker, Poly, Rect, Text, Timer

Properties   Type, Posn, Size, Coord, Align, Border, Active, Visible, Event, Thumb, Range, Step, VScroll, HScroll, Limits, Sizeable, Dragable, BCol, CursorObj, AutoConf, Data, Attach, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, Translate, Accelerator, AcceptFiles, KeepOnClose, Redraw, MethodList, ChildList, EventList, PropList

Events   Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, GotFocus, Help, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Scroll, Select

Methods   Animate, Detach, GetFocus, GetTextSize, ShowSIP

The Scroll object provides a vertical or horizontal scrollbar that can be used as a "free-standing" object or can be "attached" to the side of its parent.

An "attached" scrollbar is one that extends along one edge of a Form, SubForm or Group and has a standard width or height. When the Form or Group is resized, a vertical attached scrollbar is resized vertically but remains the same width and stays fixed to the side of its parent. Similarly, a horizontal attached scrollbar is resized horizontally but remains the same height.

For most purposes, the use of the Scroll object to provide attached scrollbars in a Form has been superseded by the provision of scrollbars as a **property** of a Form. This facility was not available in the first release of Dyalog APL/W (Version 6.2).

A "free-standing" scrollbar is typically used as a "scale" for selecting a numeric value from a range and may appear and behave rather differently from a standard attached scrollbar. Firstly, a free-standing scrollbar will normally be positioned at an arbitrary position within its parent Form or Group and be associated with other objects such as Labels and Edit fields. Secondly, when its parent Form or Group is resized, it is probably desirable that the scrollbar reacts in the same way as the other child objects, so that the overall appearance of the layout is maintained.

The Align property determines whether or not a scrollbar is attached, and if so, to which side of the parent Group or Form it is fixed. The direction of the scrollbar is determined by the VScroll and HScroll properties, which are mutually exclusive. The position and size of the scrollbar are determined by Posn and Size.

To obtain an "attached" scrollbar, it is sufficient for most purposes to specify only the Align property. If so, the direction of the scrollbar and its position and size (which are otherwise defined by VScroll, HScroll, Posn and Size) are determined automatically for you.

To obtain a "free-standing" scrollbar, it is recommended for most purposes that you set Align to '*None*' and define the orientation, position and size of the scrollbar explicitly using VScroll or HScroll, Posn and Size.

If you do attach a "free-standing" scrollbar to a particular side of its parent using Align, it will maintain its physical position (in pixels) relative to the side to which it is attached, and its dimension in that direction will remain fixed.

The Align property is a character vector containing '*Top*', '*Bottom*', '*Right*', '*Left*' or '*None*'. If you specify Align '*Right*' you get a vertical scrollbar attached to the right-hand edge of the parent Form or Group. Align '*Left*' also produces a vertical scrollbar, but one that is attached to the left-hand edge. Align '*Top*' and '*Bottom*' each produce horizontal scrollbars, attached respectively to the top and bottom edges of the Form or Group.

Note that the default value of Align is '*Right*' unless HScroll is set to ¯1 in which case it is '*Bottom*'. It must therefore be explicitly set to '*None*' if you want a non-attached "free-standing" scrollbar.

VScroll and HScroll are used to specify the orientation of the scrollbar explicitly, usually in conjunction with Align set to '*None*'. VScroll or HScroll may be specified when the object is created by □*WC*, but cannot be changed using □*WS*. The two properties are mutually exclusive. Each of them may be set to 0 or ¯1, where ¯1 means "true" and 0 means "false". Thus (VScroll ¯1) defines a vertical scrollbar, while (HScroll ¯1) specifies a horizontal one. Setting either property to ¯1 automatically causes the other to be set to 0. If you try to set both to ¯1, VScroll takes precedence and HScroll is reset to 0.

Scrolling is controlled by the Thumb, Range and Step properties.

Thumb sets and reports the current position of the "thumb" as an integer in the range 1 to the value of the Range property.

Step determines the size of changes reported when the user clicks a scroll arrow (small change) or clicks on the body of the scrollbar (large change). Step is a 2-element numeric vector whose first element specifies the value of the "small change" and whose second element specifies the value of the "large change".

Examples of attached scrollbars

```
'F' ⎕WC 'Form' 'Default ScrollBar'
```

```
'F.SCR' ⎕WC 'Scroll'
```



```
'F' ⎕WC 'Form' 'Default Horizontal ScrollBar'
```

```
'F.SCR' ⎕WC 'Scroll' ('HScroll' ¯1)
```
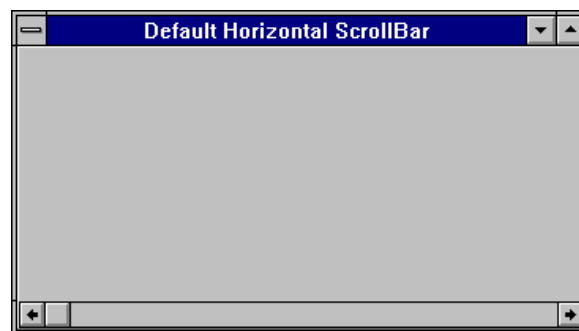
Examples of Free-Standing Scrollbars

```
'F' ⎕WC 'Form' 'Non-Default ScrollBar'

'F.SCR' ⎕WC 'Scroll' (5 45)(90 10)
             ('Align' 'None')('VScroll' ¯1)
```
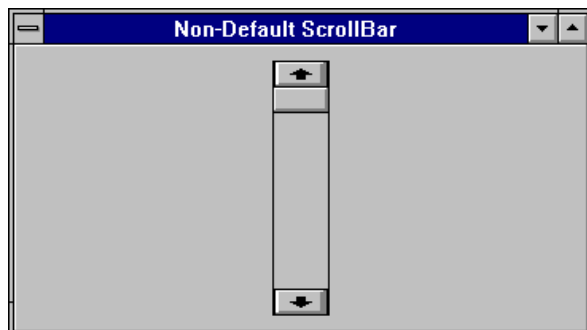


```
'F' ⎕WC 'Form' 'Horizontal ScrollBars'

'F.SC1' ⎕WC 'Scroll' (15 15)(15 70)
             ('Align' 'None')('HScroll' ¯1)

'F.SC2' ⎕WC 'Scroll' (40 40)(20 20)
             ('Align' 'None')('HScroll' ¯1)

'F.SC3' ⎕WC 'Scroll' (85 5)(10 90)
             ('Align' 'None')('HScroll' ¯1)
```
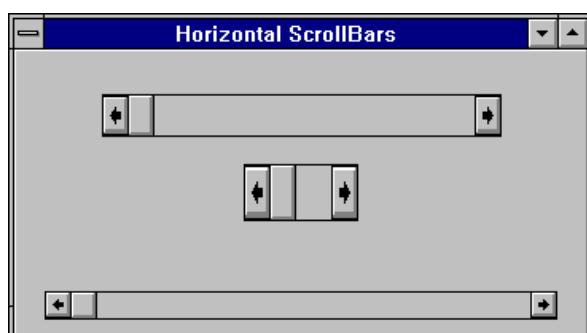
Scroll   Event 37

Applies to   Scroll, TrackBar

If enabled, this event is generated when the user attempts to move the thumb in a Scroll or TrackBar object. This can be done in one of three ways :

a)   dragging the thumb.

b)   clicking in one of the "arrow" buttons situated at the ends of the scrollbar. This is termed a small change, the size of which is defined by Step[1].

c)   clicking in the body of the scrollbar. This is termed a large change, the size of which is defined by Step[2].

The event message reported as the result of $\Box DQ$, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1] Object: ref or character vector

[2] Event code: *'Scroll'* or 37

[3] Scroll Type: numeric

[4] Position: numeric

The value of Scroll Type is 0 (drag), 1 or ¯1 (small change) or 2 or ¯2 (large change). The sign indicates the direction.

The value of Position is the new (requested) position of the thumb. Notice however, that the event is generated **before** the thumb is actually moved. If your callback function returns a scalar 0, the position of the thumb will remain unaltered.

ScrollOpposite Property

Applies to TabControl

The ScrollOpposite property specifies that unneeded tabs scroll to the opposite side of a TabControl, when a tab is selected.

ScrollOpposite is a single number with the value 0 (normal scrolling) or 1 (scrolling to the opposite side); the default is 0.

The picture below illustrates a TabControl with ScrollOpposite set to 1, after the user has clicked *Third Tab*.
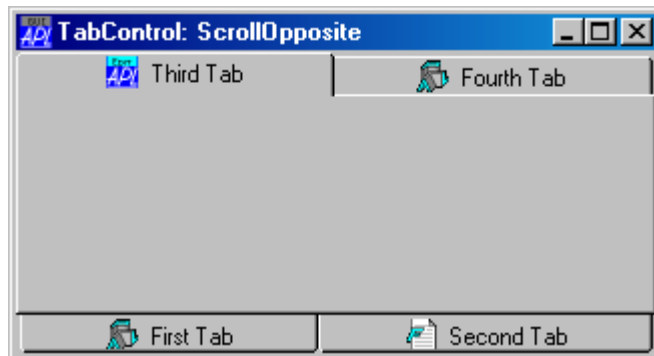
Setting ScrollOpposite to 1 implies that MultiLine is also 1.

If you set ScrollOpposite to 1 in a `⎕WC` statement, the MultiLine property will automatically be set to 1, even if you try to set MultiLine to 0 in the same statement.

If you subsequently change MultiLine back to 0 using `⎕WS` , this will work, but the effect is not useful and it is not supported.

SelDate   Property

Applies to   Calendar

The SelDate property identifies the range of dates that is currently selected in a Calendar object.

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

SelDateChange   Event 265

Applies to   Calendar

If enabled, this event is reported when the user changes the date, or range of dates, that is selected in a Calendar object. This event is also reported when the Calendar object is scrolled and the selection changes automatically to another month.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to ¯1 or by returning 0 from a callback function.

The event message reported as the result of ⎕*DQ*, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1] Object:		ref or character vector
[2] Event name or code:		'*SelDateChange*' or 265
[3] First Date:		an integer (IDN)
[4] Last Date:		an integer (IDN)

Select   Event 30

Applies to   ActiveXControl, Bitmap, Button, Calendar, Circle, Clipboard, Combo, ComboEx, Cursor, DateTimePicker, Edit, Ellipse, FileBox, Font, Form, Grid, Group, Icon, Image, Label, List, ListView, Locator, Marker, MDIClient, Menu, MenuItem, Metafile, Poly, Printer, ProgressBar, Rect, RichEdit, Scroll, Spinner, Static, StatusBar, StatusField, SubForm, TabBar, TabBtn, TabButton, Text, ToolBar, ToolButton, TrackBar, TreeView, UpDown

For a Button with Style *'Push'* this event is generated when the user "pushes" the button. This can be done by clicking the left mouse button, or by pressing the Enter key or the space bar when the Button has the focus. The Select event can also be generated when the Button does not have the focus, by pressing the Enter key when its Default property is 1 or by pressing the ESC key when its Cancel property is 1.

For a Button with Style *'Radio'* or *'Check'* this event is generated when the user toggles the button from one state to another. This can be achieved by clicking the left mouse button or by pressing the space bar when the Button has the focus.

For a Combo or List object, a Select event is generated when the user selects an item from the list, whether by pressing the arrow keys or by clicking the left mouse button.

For a MenuItem, a Select event is generated when the user chooses the item.

For all other objects, this event is generated when the user presses the keys associated with the object's Accelerator property.

The event message reported as the result of *⎕DQ*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1] Object: ref or character vector

[2] Event code: *'Select'* or 30

SelImageIndex   Property

Applies to   ComboEx, TreeView

The SelImageIndex property determines which bitmapped images in an ImageList correspond to items in a ComboEx or TreeView object when the item is *selected*. It is an integer vector whose length is the same as the number of items in the object and is $\square IO$ dependent.

See also ImageIndex

SelItems   Property

Applies to   Combo, ComboEx, Grid, List, ListView, TreeView

This property determines which (if any) of the items in an object are currently selected and highlighted. Except for a Grid, it is a Boolean vector with one element per item in the list. A value of 1 means "selected"; 0 means "not selected".

This property is used after a Select event to identify which item has been chosen. In a Combo or a List with Style *'Single'* only one element will have the value 1.

SelItems should also be used to pre-set the contents of the edit field in a Combo box with Style *'Drop'*. In Combo boxes with Style *'Simple'* or *'DropEdit'*, the contents of the edit field may also be specified by the Text property. If you specify both, the value of Text takes precedence.

In a Grid SelItems is a 2-element vector of 2-element integer vectors that identifies the row and column co-ordinates of the first and last cells in the currently selected block of cells. If multiple selection is enabled, SelItems may be a vector of such arrays, specifying the coordinates of a number of non-contiguous blocks of selected cells.

SelRange   Property

Applies to   TrackBar

The SelRange property specifies the selected range in a TrackBar which has Style *'Selection'*. It is a 2-element numeric vector.

SelText   Property

Applies to   Combo, ComboEx, Edit, RichEdit

This property determines or identifies the portion of text in an object that is currently selected and highlighted. It can be used to pre-select all or part of the text to be replaced or deleted when the user starts typing. It can also be used to query the area of text that the user has highlighted. This can be useful if you want to implement your own cut/paste/replace features.

SelText is always a 2-element integer vector. If the field contents (defined by the Text property) is a vector, SelText is simple. Its first element is the index of the first selected character and its second element is 1 + the index of the last selected character. The length of the selected string is therefore obtained by subtracting the first element from the second.

If there are no characters selected, the two elements are equal and specify the current position of the input cursor.

If the contents is a vector of vectors or a matrix, each element of SelText is a 2-element vector. The first item in each of the elements indexes the vector (in a vector of vectors) or row (in a matrix). The second item in each element indexes the position of the character in the vector or along the row. Again, the value reported for the last character in the selected string is 1 + its index.

Separator   Object

Purpose	A horizontal or vertical line used to separate items in a menu.
Parents	Menu, MenuBar
Children	Timer
Properties	Type, Posn, Style, Event, FCol, BCol, Data, EdgeStyle, Translate, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create
Methods	Detach

This object provides a vertical or horizontal line to separate items in a Menu. It may also be used to split a MenuBar over more than one line.

The orientation of the Separator is determined by its Style property, which may be `'Horz'` (horizontal) or `'Vert'` (vertical). The default is `'Horz'`.

If you want to provide a menu with a 3-Dimensional (pushbutton) appearance, you should also set the EdgeStyle property on any Separator objects in it. Alternatively, you can achieve the same effect by setting the background colour (BCol) for the Separators to grey (192 192 192).

The Posn property is a single integer number which specifies the positional index of the Separator relative to the other objects in the Menu. A Separator does not generate any events.

Like other components of a menu, the position of a Separator is normally determined by the order in which it is created in relation to other objects with the same parent. However, you can use the Posn property to **insert** a Separator into an existing structure. For example, having defined three MenuItem objects as children of a Menu, you can insert a Separator between the first and the second by specifying its Posn to be 2. Note that the value of Posn for the MenuItems that were previously second and third will then be reset to 3 and 4 respectively.

If you put a Separator (either Style) into a MenuBar, it has the effect of adding another line to it. Any items added after the Separator will appear in the new line.

ServerVersion   Property

Applies to   OLEServer

This property specifies the version number of an OLEServer object.

It is a 2-element integer vector that specifies the major and minor version numbers respectively.

The default value of ServerVersion is (`1 0`).

SetCellSet   Method 171

Applies to   Grid

The SetCellSet method sets the value of the CellSet property of a Grid for a particular cell.

The argument to SetCellSet is a 3-element array as follows:

[1] Row:	integer
[2] Column:	integer
[3] Value:	0 or 1

SetCellType   Method 156

Applies to   Grid

This method is used to change the type of a particular cell in a Grid.

The argument to SetCellType is a 3-element vector as follows :

      [1] Cell row: integer

      [2] Cell column: integer

      [3] Cell type: integer

SetColSize   Event 176

Applies to   Grid, ListView

If enabled, this event is reported when the user changes the width of a column in a Grid or ListView object, or changes the width of the row titles in a Grid. This may be done by dragging a border with the mouse or (in a Grid) by double-clicking over a border. In the former case, the default action is to adjust the width of the appropriate column or the width of the row title area to the size selected by the user. In the latter case, the default action is to adjust the width to the maximum required to display all the data and column title.

In either case, you can disable the default action by setting the event action code to ¯1 or you can selectively prevent a particular resize operation from taking place by returning 0 from a callback function.

The event message reported as the result of ⎕DQ, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1] Object:		ref or character vector
[2] Event name or code:		*'SetColSize'* or 176
[3] Column number:		Integer. This is sensitive to the index origin, ⎕IO, but is ¯1 if the user has resized the row titles in a Grid.
[4] Width:		Integer containing the value of the (new) column width. This is ¯3 if the user has double-clicked to request automatic width adjustment.
[5] Undo flag :		0 or 1

You can resize a column or resize the row titles under program control by calling SetColSize as a method. If for a Grid, you specify ¯1 as the *Width* parameter the column will be resized to its default width. If you specify a value of ¯2 the column will be resized to fit the data, but the width of the column title is ignored. A value of ¯3 resizes the column to fit the data *and* the column title (whichever is the greater). The Undo flag is applicable only to a Grid object and is always 1 if the event was generated by the user.

The following expression will size the first *NCOLS* columns of a Grid called *F.G* to fit the data and the column titles:

```
{F.G.SetColSize ω ¯3}¨ιNCOLS
```

SetEventInfo   Method 547

Applies to   ActiveXControl, OLEServer

This method is used to register an event that may be generated by an ActiveXControl or OLEServer object.

A host application that wishes to attach a callback function to an event in a Dyalog APL ActiveXControl or OLEServer, needs to know the name of the event and the number and data types of any parameters that the event may supply. It also needs to know the data type (if any) of the result that the callback function may be expected to pass back to the control.

An ActiveXControl or OLEServer generates an event in the host application using 4 $\Box NQ$. The right argument is a vector whose first 2 elements are character vectors containing the names of the ActiveXControl or OLEServer and the event respectively. The parameters for the event are passed as additional elements in the argument.

Another way to think about it is that when you generate an event using 4 $\Box NQ$, you are effectively calling a function, of your specification, in the host application. To enable the host application to accept the function call, it needs to know the number of parameters that you will supply and their data types.

A further consideration is that if you specify that the data type of a parameter is a *pointer* (e.g. `'VT_PTR TO I4'`) this will allow a callback function to modify the parameter in-situ. If so, the result returned by 4 $\Box NQ$ will be the modified values of any such parameters; this is a similar mechanism to $\Box NA$.

The argument to SetEventInfo is a 1, 2 or 3-element array as follows:

       [1] Event name: character vector

       [2] Event info: nested array (see below)

       [3] Help ID: integer

Event Info

Event info, specifies an optional help string which describes what the event does, the data type of the result (if any) and the names and data types of its arguments.

If the event is fully described, each element of Event Info is a 2-element vector of character vectors. The first element contains the help string and the COM data type of the result that the callback function in the host application is expected to supply. Subsequent elements contain the name and COM data type of each of the parameters supplied by the event.

However, both the help string and the names of the parameters are optional and may be omitted. If so, one or more elements of *Event Info* may be a simple character vector.

Help ID

This is an integer value that identifies the help context id for the event within the help file associated with the HelpFile property of the ActiveXControl object. The value ¯1 means that no help is provided. APL stores this information in the registry from where it may be retrieved by the host application.

Example

The example *Dual* ActiveXControl, that is fully described elsewhere, generates a ChangeValue1 event. This event occurs when the user moves the thumb in a TrackBar that is internal to an instance of the ActiveXControl.

The external ChangeValue1 event is fired by an internal APL callback function (called *ChangeValue*) that is attached to ThumbDrag and Scroll events on the TrackBar object. The internal callback function is :

```
[0]    ChangeValue MSG
[1]    ⍝ Callback for ThumbDrag and Scroll
[2]    Value1←⊃4 □NQ'' 'ChangeValue1'(⊃¯1↑MSG)
[3]    CalcValue2
[4]    'V1'□WS'Text'(⍕Value1)
[5]    'V2'□WS'Text'(⍕Value2)
```

Note that *ChangeValue*[2] generates the external ChangeValue1 event by invoking 4 □NQ, passing it the new value provided by the TrackBar. However, the host application is permitted to modify that value, returning it in the result of 4 □NQ. This result, rather than the TrackBar value itself, is then used to update other (Label) controls in the object.

The following statements were used to declare the *ChangeValue1* event The event provides a single parameter named *Value1* that may be modified in-situ by a callback function in the host application. The callback is not, otherwise, expected to return a result.

```
INFO←⊂'Occurs when value of control is changed' 'VT_VOID'
INFO,←⊂'Value1' 'VT_PTR TO VT_I4'
F.Dual.SetEventInfo 'ChangeValue' INFO
```

If the host application was Visual Basic, a suitable callback function might be:

```
Private Sub Dual1_ChangeValue1(Value1 As Long)
Value1=2*(Value1\2)
End Sub
```

This callback function receives the proposed new value of the control as the parameter Value1, and modifies it, forcing it to be an even number.

SetFinishText   Method 366

Applies to   PropertySheet

The SetFinishText method sets the caption of the Finish button in a Wizard-style PropertySheet.

The argument to SetFinishText is a single item as follows:

      [1] Finish button text: character vector

SetFnInfo Method 545

Applies to ActiveXControl, OLEServer

This method is used to describe an APL function that is to be exported as a method, or as a property get or property put function, of an ActiveXControl or OLEServer object.

An exported function must be a niladic or monadic defined function (dynamic functions and derived functions are not allowed) and may optionally return a result. Ambivalent functions (functions with optional left argument) are allowed, but will be called monadically by the host application.

COM syntax differs from APL syntax in many ways and the SetFnInfo method is required to declare an APL function to COM in terms that COM understands. In particular, although monadic APL functions take just one argument, COM functions may take several parameters, and some may be optional.

A function exported by SetFnInfo will be called by a host application with the number of parameters that SetFnInfo has described. The argument received when the function is called by a host application, will be a nested vector of this length.

The argument to SetFnInfo is a 2, 3 or 4-element array as follows:

[1] Function name:	character vector
[2] Function info:	nested array (see below)
[3] Help ID:	integer
[4] Function type:	integer
[5] Property name:	character vector

Function Info

Function Info, specifies an optional help string which describes what the function does, the data type of the result (if any) and the names and data types of its arguments.

If the function syntax is fully described, each element of *Function Info* is a 2-element vector of character vectors. The first element contains the help string and the COM data type of the function's result. Subsequent elements contain the name and COM data type of each parameter.

However, both the help string and the names of the parameters are optional and may be omitted. If so, one or more elements of *Function Info* may be a simple character vector.

Consider a very basic function *ADD* in an ActiveXControl called *F.dbase*, that is designed to add a record to a personnel database. The database consists only of a list of names, ages and addresses.

Function *ADD* expects to be called with a name (character string), age (number) and address (character string), and returns a result 0 or 1 (Boolean) according to whether the record was successfully added. This function could be declared as follows:

```
HELP←'Adds a new record to the personnel database'
SPEC←⊂(HELP 'VT_BOOL')      ⍝ Result is Boolean
SPEC,←⊂('Name' 'VT_BSTR')   ⍝ 1st param 'Name' is string
SPEC,←⊂('Age' 'VT_I4')      ⍝ 2nd param 'Age' is integer
SPEC,←⊂('Address' 'VT_BSTR')⍝ 3rd param 'Address' is string

F.dbase.SetFnInfo 'ADD' SPEC
```

Alternatively, but much less helpfully, the function could be declared to take a single unnamed nested argument, leaving it to the host application programmer to guess at its structure :

```
SPEC←⊂('' 'VT_BOOL')   ⍝ No help string, result is Boolean
SPEC,←⊂('' 'VT_ARRAY OF VT_VARIANT') ⍝ Param is nested

F.dbase.SetFnInfo 'ADD' SPEC
```

Help ID

This is an integer value that identifies the help context id within the help file associated with the HelpFile property of the ActiveXControl object. The value ¯1 means that no help is provided. APL stores this information in the registry from where it may be retrieved by the host application.

Function Type

This specifies the type of function being exported. This is an integer with one of the following values:

1	Function is a *method*
2	Function is a *property get* function
4	Function is a *property put* function

In both these last two cases, the name of the property, which is totally independent of the name of the APL function, is given as the *Property name* Parameter.

If omitted, the function type is *method*.

Definitions

A *method* is function that may be called directly by a host application.

A *property get* function is a function that is invoked by OLE when a host application references a specific property.

Property get and property put functions provide an alternative to representing a property as a variable.

A property get function allows your ActiveXControl to derive the current value of a property dynamically, rather than having to continually keep it updated in a variable. For example, a property such as the current contents of an Edit box within an ActiveXControl is better represented by a property get function than by a variable.

A *property put* function is a function that is invoked by OLE when a host application assigns a value to a specific property.

Property get and property put functions provide an alternative to representing a property as a variable.

A property put function allows your ActiveXControl to validate before accepting a new value assigned by the host application It also allows it to action side effects, such as updating the user interface, to reflect the new value.

SetItemImage   Method 315

Applies to   TreeView

This method is used to allocate a picture icon to a particular item in a TreeView object.

The argument to SetItemImage is a 2-element array as follows:

> [1] Item number: Integer.
>
> [2] Picture index: Integer.

Item number is the index of the item concerned.

Picture index is an index into the array of bitmapped images in the corresponding ImageList object which is referenced via the ImageListObj property.

SetItemPosition Event 322

Applies to ListView

If enabled, this event is reported when the user drag-drops an item within a ListView object. This operation may be disabled by returning 0 from a callback function.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 7-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	`'SetItemPosition'` or 322
[3] Item number:	Integer. The index of the item.
[4] Y-position:	Integer. New y-position of the item.
[5] X-position:	Integer. New x-position of the item.
[6] Button number:	Integer. The mouse button used to perform the drag.
[7] Shift State:	Integer: Sum of shift key codes (number) 1 = Shift key is down 2 = Ctrl key is down 4 = Alt key is down

SetItemState   Method 307

Applies to   ListView, TreeView

This method is used to set the status of a particular item in a ListView or TreeView object.

The argument to SetItemState is a 2-element array as follows:

      [1] Item number: Integer. The index of the item concerned.

      [2] Status: Integer

The status of an item is calculated as the sum of one or more of the following state codes:

1	Item has the focus
2	Item is selected
8	Item is highlighted for dropping
16	Item is displayed in bold text
32	Item is expanded
64	Item is or has been expanded
4096	Item is checked (see CheckBoxes)

SetMethodInfo   Method 546

Applies to   OCXClass, OLEClient

This method is used to redefine the arguments or data types associated with a method that is exported by a COM object. SetMethodInfo is used to override the information provided by the object's Type Library.

The argument to SetMethodInfo is a 2 or 3-element array as follows:

> [1] Method name: character vector
>
> [2] Method info: nested vector (see below)
>
> [3] Method index: integer

If you wish to describe the method completely, the structure of *Method info* should be identical to the structure returned by GetMethodInfo, although abbreviated formats are also allowed.

If the object exports the method directly, and not through the standard IDispatch interface, you must also specify *Method Index*, which is the index of the method in the object's virtual table (vtable). This information may be available in printed documentation or in a C-language header file.

For example, the InchesToPoint method exported by *Excel.Application* takes a single argument whose name is *Arg1* and whose data type is VT_R8. The function returns a result of the same data type. The details provided in the *Excel.Application* Type Library are in fact correct, but if you wanted to redefine them, the following statements could be used to describe the InchesToPoints method.

```
methodinfo← ('' 'VT_R8')('Arg1' 'VT_R8')
EX.SetMethodInfo 'InchesToPoints' methodinfo
```

Note that the structure of variable *methodinfo* is identical to the result of the GetMethodInfo method.

```
        DISPLAY methodinfo
.→----------------------------------.
| .→------------.  .→--------------. |
| | .⊖. .→----. |  | .→---. .→----. | |
| | | | |VT_R8| |  | |Arg1| |VT_R8| | |
| | '-' '-----' |  | '----' '-----' | |
| '∈------------'  '∈--------------' |
'∈----------------------------------'


        DISPLAY EX.GetMethodInfo 'InchesToPoints'
.→----------------------------------.
| .→------------.  .→--------------. |
| | .⊖. .→----. |  | .→---. .→----. | |
| | | | |VT_R8| |  | |Arg1| |VT_R8| | |
| | '-' '-----' |  | '----' '------' | |
| '∈------------'  '∈--------------' |
'∈----------------------------------'
```

Unless you are going to call the method using the names of its arguments, these names are clearly superfluous and may be omitted, for example:

```
        methodinfo← 'VT_R8' 'VT_R8'
        EX.SetMethodInfo 'InchesToPoints' methodinfo
```

SetPropertyInfo   Method 554

Applies to   OCXClass, OLEClient

This method is used to redefine a property that is exported by a COM object.
SetPropertyInfo is used to override the information provided by the object's Type
Library.

The argument to SetPropertyInfo is a 2 or 3-element array as follows:

> [1] Property name: character vector

> [2] Property info: nested vector

> [3] Property function: integer

For example, the Visible property exported by *Excel.Application* has the data type
VT_BOOL and may be declared as follows:

```
'EX' ⎕WC 'OLEClient' 'Excel.Application'
EX.SetPropertyInfo 'Visible' 'VT_BOOL'
```

Property function may be required if the property value is retrieved or set via a function.
This typically applies if the property takes parameters and will result in the property
being fixed as a function rather than as a variable. Such properties may have a
PropertyGet function, a PropertyPut function and/or a PropertyPutByReference
function. If so, it is necessary to say to which of these three functions the details apply.
The value of *Property function* is an integer 2 (PropertyGet), 4 (PropertyPut), or 8
(PropertyPutByReference).

For example, the following statement declares the PropertyGet function for the Item
property of the Fields collection of the OLE object DAO.DBEngine. This property takes
an index (into the collection) and returns an object.

```
Fields.SetPropertyInfo 'Item'('VT_DISPATCH' 'VT_I4')2
```

SetRowSize   Event 175

Applies to   Grid

If enabled, this event is reported when the user changes the height of a row or changes the height of the column titles. This may be done by dragging a border with the mouse or by double-clicking over a border. In the former case, the default action is to adjust the height of the appropriate row or the height of the column title area to the size selected by the user. In the latter case, the default action is to adjust the height to the maximum required to display all the data.

In either case, you can disable the default action by setting the event action code to ‾1 or you can selectively prevent a particular resize operation from taking place by returning 0 from a callback function.

The event message reported as the result of ⎕*DQ*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	'*SetRowSize*' or 175
[3] Row number:	Integer. This is sensitive to the index origin, ⎕*IO*, but is ‾1 if the user has resized the column titles.
[4] Height:	Integer containing the value of the (new) row height. This is ‾3 if the user has double-clicked to request automatic height adjustment.
[5] Undo flag :	0 or 1

You can resize a row or resize the column titles under program control by calling SetRowSize as a method. If you specify ‾1 as the *Height* parameter, the row will be resized to its default height .. If you specify a value of ‾2 the row will be resized to fit the data. The following expression will set the heights of first *NROWS* rows of a Grid called *F.G* to fit the data and the row titles.

```
{F.G.SetRowSize ⍵ ‾3}¨⍳NROWS
```

The Undo flag is always 1 if the event was generated by the user.

SetSpinnerText Event 421

Applies to Spinner

If enabled, this event is generated when the user clicks one of the spin buttons in a Spinner object. The event is reported *after* the value of the Thumb property has been updated but *before* the Text property has been changed. You may use this event to set the text in the Spinner dynamically instead of relying on it being updated automatically.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	`'SetSpinnerText'` or 421
[3] Thumb value:	Integer. The new value of the Thumb property resulting from the user pressing one of the spin buttons.
[4] Text:	The text that is about to be put into the edit field.

The SetSpinnerText event is designed to allow you to dynamically set the text in the Spinner in response to a spin button. It might be used in circumstances where the set of items you wish to present to your user is not predictable in advance.

Setup Method 101

Applies to Printer

This method causes the system to display a standard Printer Setup dialog box to be displayed to allow the user to alter the printer settings. This is a "modal" dialog box that must be closed before the APL application can continue.

The Setup method is niladic.

If you attach a callback function to this event and have it return a value of 0, the dialog box will not appear.

SetVarInfo   Method 546

Applies to   ActiveXControl, OLEServer

This method is used to describe an APL variable that is to be exported as a property of an ActiveXControl or OLEServer object.

The argument to SetVarInfo is a 2 or 3-element array as follows:

[1] Variable name:	character vector
[2] Variable info:	see below
[3] Help ID:	integer

Variable info is either a simple character vector that specifies the COM data type of the variable, or a 2-element vector of character vectors whose first element specifies a help string and whose second element specifies the COM data type.

Help ID is an optional integer value that identifies the help context id within the help file associated with the HelpFile property of the ActiveXControl object. The value ¯1 means that no help is provided. APL stores this information in the registry from where it may be retrieved by the host application.

COM data type

The table below shows the correspondence between COM data types and APL arrays.

OLE DataType	APL array
VT_BOOL	numeric scalar
VT_I1	numeric scalar
VT_I2	numeric scalar
VT_I4	numeric scalar
VT_R4	numeric scalar
VT_R8	numeric scalar
VT_BSTR	character vector
VT_CY	2-element numeric vector
VT_DATE	6 element numeric vector
VT_VARIANT	any array
VT_SAFEARRAY	any array (VT_ARRAY OF VT_VARIANT)
VT_DISPATCH	$\square OR$ of a namespace
VT_COLOR	3-element RGB

APL vectors may be described by pre-fixing the data type string with
`'VT_ARRAY OF '`. For example `'VT_ARRAY OF BSTR'` specifies a vector of
character vectors.

If the APL array is the $\square OR$ of a namespace, its data type should be specified as
`'VT_DISPATCH'`.

SetWizard   Event 365

Applies to   PropertyPage

If enabled, this event is reported when the user has clicked the Next or Back button in a PropertySheet with Style `'Wizard'`. This action also generates PageNext (or PageBack) and PageDeactivate and PageActivate events. The SetWizard event is the final event to be reported as a result of this action, and is the only one that is affected by the result of a callback function. The event message reports the active/inactive state of the 3 page changing buttons (Back, Next and Finish) that should result from the action. Note that the Next and Finish buttons occupy the same position and are mutually exclusive.

The event message reported as the result of $\Box DQ$, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	`'SetWizard'` or 365
[3] Active state Back:	0 or 1
[4] Active state Next:	0 or 1
[5] Active state Finish:	0 or 1
[6] Finish caption:	character vector

You may alter the state of the buttons by changing elements [3-5] of the event message and returning it as a result of your callback. You may also set the state of the buttons at any time by calling SetWizard as a method.

When the event is reported by $\Box DQ$, element 6 is an empty vector. If you modify it and return it in the result of a callback, the caption of the Finish button changes accordingly and the Back and Next buttons disappear. This happens regardless of the states you specified in elements [3-5].

ShowCaptions Property

Applies to   ToolControl

The ShowCaptions property specifies whether or not the captions of individual ToolButton objects are drawn. ShowCaptions is a property of the parent ToolControl object.

ShowCaptions is a single number with the value 0 (ToolButton captions are not shown) or 1 (ToolButton captions **are** shown); the default is 1

ShowCaptions allows you to toggle end-user preferences for the display of ToolButton captions, without having to set/clear individual captions one by one.

ShowComment   Event 223

Applies to   Grid

If enabled, a Grid will generate a ShowComment event when the user rests the mouse pointer over a commented cell. You may use this event to modify the appearance of the comment dynamically.

The event message reported as the result of $\Box DQ$ or supplied as the right argument to your callback function is an 8-element vector containing the following:

[1]	Object:	ref or character vector
[2]	Event name or code:	*'ShowComment'* or 223
[3]	Cell row:	integer
[4]	Cell column:	integer
[5]	Comment text:	character vector
[6]	Window height:	integer, pixels
[7]	Window width:	integer, pixels
[8]	Tip behaviour flag:	(1 = yes; 0 = no)

A callback function may modify the standard behaviour. You may prevent the comment from being displayed by returning 0 as the result of the callback. Alternatively, you may modify the comment text, its window size, or its pop-up behaviour by changing the appropriate element(s) of the event message and returning the new event message as the result.

You may display the comment associated with a particular cell under program control by calling ShowComment as a method. In this case, only the *Cell row* and *Cell column* parameters need be specified. If however, you wish to override the comment text and/or its window size, you may do so (temporarily) by specifying the corresponding parameters. By default, a comment displayed under program control does not exhibit tip behaviour but remains visible until it is explicitly removed using the HideComment method.

Note that a comment will only be displayed if the specified cell is marked as a commented cell.

ShowDropDown   Property

Applies to   ColorButton, ToolControl

The ShowDropDown property specifies whether or not a drop-down menu symbol is drawn in a ColorButton or alongside ToolButton objects which have Style *'DropDown'*.

ShowDropDown is a single number with the value 0 (drop-downs are not shown) or 1 (drop-downs **are** shown); the default is 1.

ShowDropDown also affects the behaviour of ToolButton objects which have Style *'DropDown'*.

If the ShowDropDown property of the parent ToolControl is 0, clicking the ToolButton causes the popup menu to appear. In this case, the ToolButton itself does not itself generate a Select event; you must rely on the user selecting a MenuItem to specify a particular action.

If the ShowDropDown property of the parent ToolControl is 1, clicking the dropdown button causes the popup menu to appear; clicking the ToolButton itself generates a Select event, but does not display the popup menu.

The following picture illustrates a ToolControl with ShowDropDown set to 1.
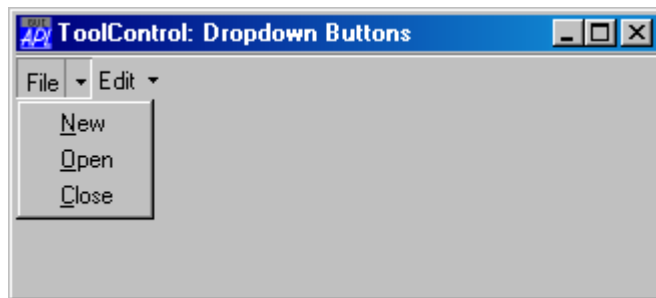
ShowHelp   Method 580

Applies to   OCXClass, OLEClient

This method is used to display the Windows help file for a COM object or the help topic associated with one of its properties, events or methods.

The argument to ShowHelp is ᶱ, or a single item as follows :

     [1] Topic: character vector.

Topic specifies the name of a property, event or method.

In the case of an OLE Control, the object name may be the name of an OCXClass or an *instance* of an OCXClass.

ShowInput   Property

Applies to   Grid

This property specifies whether or not the cells in a Grid are displayed using their associated input objects.

The ShowInput property is either a single Boolean value that applies to all the cells in a Grid, or it is a vector whose elements are mapped to individual cells via the CellTypes property. A value of 0 means that the corresponding cell is displayed normally. A value of 1 indicates that the cell is displayed using its associated input object, as it is when it is the current cell. ShowInput is relevant to cells displayed using Combo and Button objects.

The example below illustrates the appearance of a Grid in which ShowInput is set to 0 for the Job Title column and 1 for the Region and Permanent columns.

The appearance of the same Grid but with ShowInput set to 0 throughout is illustrated below:

ShowItem   Method 316

Applies to   TreeView

This method is used to display a particular item in a TreeView object.

The argument to ShowItem is a single item as follows:

> [1] Item number: Integer.

Item number specifies the index of the item concerned.

In order to display the requested item, the parent item (if any) will be opened and the object will be scrolled if necessary.

ShowProperties   Method 560

Applies to   OCXClass

This method is used to display the PropertySheet for *an instance of* an OLE Control. The user may then modify some or all of the properties of the control by changing values in the property sheet. This facility is intended to be used in the context of a GUI design tool but may also be useful in certain end-user applications.

The ShowProperties method is niladic.

ShowSession   Property

Applies to   OLEServer

This property specifies whether or not the APL Session window is displayed when an OLEServer object is started by an OLE client.

Its default value is 0 (do not display the Session).

Note that if RunMode is `'MultiUse'`, you may not in any way access the *instances* of the object that are being controlled by the client applications, even if only a single client is connected.

ShowSIP   Method 25

Applies to   ActiveXControl, Animation, Button, Calendar, ColorButton, Combo, ComboEx, CoolBar, DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, ProgressBar, PropertyPage, RichEdit, Root, Scroll, SM, Spinner, Static, StatusBar, SubForm, TabBar, TabControl, ToolBar, ToolControl, TrackBar, TreeView, UpDown

ShowSIP applies only to Pocket APL. In versions of Dyalog APL for other platforms, it has no effect.

This method displays and hides the Input Panel.

The argument to ShowSIP is 1 (display the Input Panel) or 0 (hide the Input Panel).

The argument to ShowSIP is 0 or 1 as follows :

      [1] Mode: Boolean
                                          0 = hide the Input Panel.
                                          1 = display the Input Panel

The result of ShowSIP is 1 if the Input Panel was previously displayed, or 0 if it was previously hidden.

ShowThumb Property

Applies to TrackBar

The ShowThumb property specifies whether or not the thumb in a TrackBar object is visible. It is Boolean with a default value of 1 and it may be toggled on and off using ⎕ws.

SingleClickExpand Property

Applies to TreeView

The SingleClickExpand property specifies whether or not an item in a TreeView control is expanded when the user selects the item.

SingleClickExpand is a single number with the value 0 (the user must select the *expand icon* to cause the item to expand) or 1 (the item is expanded when the *text* of the item is selected); the default is 0.

SIPMode   Property

Applies to   Form

SIPMode applies only to Pocket APL. In versions of Dyalog APL for other platforms, it has no effect.

This is a Boolean property that specifies the behaviour of the Input Panel with respect to the Pocket APL GUI.

If SIPMode is 1, the Input Panel is automatically displayed when a GUI control that may receive character input (e.g. an Edit object) receives the input focus. The Input Panel is automatically hidden when the input focus moves to a control that does not receive character input.

If SIPMode is 0 (the default), the display of the Input Panel is not handled automatically, but may be controlled using the ShowSIP method.

Note that the user may display and hide the Input Panel manually, regardless of the value of SIPMode.

SIPResize   Property

Applies to   Form

SIPResize applies only to Pocket APL. In versions of Dyalog APL for other platforms, it has no effect.

This is a Boolean property that specifies the behaviour of a Form when the Input Panel is raised or lowered.

If SIPResize is 1 (the default), the Form generates a Configure event when the Input Panel is raised or lowered. Unless disabled or modified by a callback function, the Form is automatically resized to occupy the entire space above the Input Panel.

If SIPResize is 0, the Form does not generate a Configure event when the Input Panel is raised or lowered. This means that, at times, the lower part of the Form may be obscured by the Input Panel.

Size Property

Applies to   ActiveXControl, Animation, Bitmap, Button, Calendar, ColorButton, Combo, ComboEx, CoolBand, CoolBar, DateTimePicker, Edit, Ellipse, Font, Form, Grid, Group, Image, ImageList, Label, List, ListView, Locator, Marker, MDIClient, Metafile, ProgressBar, PropertyPage, PropertySheet, Rect, RichEdit, Root, Scroll, SM, Spinner, Splitter, Static, StatusBar, StatusField, SubForm, TabBar, TabBtn, TabButton, TabControl, ToolBar, ToolButton, ToolControl, TrackBar, TreeView, UpDown

This is a 2-element numeric vector specifying the height and width of the object.

For the Bitmap object, Size is set and reported in pixels. Setting the Size of a Bitmap causes it to be scaled (up or down).

For all other objects, Size is reported and set in units defined by the Coord property and, if Coord is `'User'`, the XRange and YRange properties of the object's parent.

For the Root object, if Coord is `'Prop'` the value of Size is (100,100). If Coord is `'Pixel'` the value of Size reports the number of pixels on the screen.

For a Form or SubForm, the Size property defines the area within the object, and excludes its title bar, menu bar and border if these are present.

For a Combo object with a "drop-down" list, the first element of Size (height) is ignored. The height of the edit field is determined by the height of the font, while the size of the list box is determined by the Rows property.

For a Metafile object, Size specifies the *granularity* of the Metafile and defaults to the size of its parent.

When specifying Size, you can set the height or width to a default value (□WC) or leave it unchanged (□WS) by giving the corresponding element a value of 0.

Sizeable   Property

Applies to   Animation, Button, Calendar, ColorButton, Combo, ComboEx, DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView, Locator, ProgressBar, RichEdit, Scroll, SM, Spinner, Static, StatusBar, StatusField, SubForm, TabBar, ToolBar, TrackBar, TreeView, UpDown

This property determines whether or not an object can be directly resized by the user once it has been created by `⎕WC`.

It is a single number with the value 0 (the object cannot be resized by the user) or 1 (the object may be resized by the user). The default is 1.

For a Form or SubForm, the Sizeable property may only be set by `⎕WC` and cannot subsequently be altered using `⎕WS`. An attempt to do so generates a *NONCE ERROR*. For a Form, the default value is 1 and the Form is a standard resizable window with a border. Note that the value of Sizeable is independent of the values of the MaxButton and MinButton properties, so that a Form with MaxButton 1 can be maximised even though its Sizeable property is 0.

For other objects, the default value of the Sizeable property is 0. However, setting it to 1 (which may be done dynamically using `⎕WS`) allows the user to resize it with the mouse.

In all these cases, when the user resizes an object, the object will generate a Configure (31) event.

Sizeable also applies to the Locator object. In this case, a value of 1 implies "rubberbanding" and a value of 0 means "no rubberbanding". See Locator object for further details.

SM	Object

Purpose   Defines a window for ⎕*SM*/⎕*SR*.

Parents   Form, Group, PropertyPage, SubForm, ToolBar, ToolControl

Children   Cursor, Timer

Properties   Type, Posn, Size, Coord, Border, Visible, Event, Sizeable, Dragable, BCol, Picture, CursorObj, AutoConf, Data, Attach, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, AcceptFiles, KeepOnClose, Redraw, MethodList, ChildList, EventList, PropList

Events   Close, Configure, Create, DragDrop, Help, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseUp

Methods   Animate, Detach, GetFocus, GetTextSize, ShowSIP

This object defines a window for ⎕*SM*/⎕*SR* and allows you to combine the functionality of ⎕*SM*/⎕*SR* with the "windows" GUI. For example, you can define a Form with a MenuBar at the top and a ⎕*SM* window beneath it, with perhaps some Buttons alongside.

To allow the user to interact with both ⎕*SM* and other top-level objects, you must specify the names of these objects in the right argument of ⎕*SR*. Thus the statement :

```
CTX ← KEYS CTX ⎕SR 1 2 3 'Form1'
```

allows the user to interact with fields (rows) 1-3 of ⎕*SM* **and** with the object '*Form1*' and its children. Callback functions associated with events in '*Form1*' will be executed automatically by ⎕*SR*. If an enabled event without a callback occurs, the event will be placed on ⎕*DQ*'s internal queue and ⎕*SR* will terminate. The nature of the termination (i.e. that it was caused by an event in an object) is reported by the value 131072 (2*17) in the fourth element of ⎕*SR*'s result. The specific event (Configure, MouseUp, etc.) is however not reported. It is therefore generally preferable to use callbacks.

The Posn, Size and Coord properties allow you to specify the position and size of the window occupied by ⎕*SM* within its parent Form. Note however that the ⎕*SM* window will automatically be sized to be an exact number of characters in height and width which will be reported in ⎕*SD*.

The Border property may be used to specify a border around the outside of the □*SM* window. It is a number with the value 0 (no border) or 1 (1 pixel border). The default is 0. The EdgeStyle property may be used to give the object a 3-dimensional appearance. Its default value is '*Recess*'. The area within the SM object that is defined by □*SM* is necessarily a multiple of the character size. The region between this area and the outer edges of the object is coloured by the background colour specified by BCol, or may be filled with a bitmap specified by Picture.

If the user resizes the Form which contains the SM object, the SM object will generate a Configure event if enabled. If the Configure event is not enabled, □*SR* will terminate with a RESIZE error which can be trapped using □*TRAP*. Either method can be used to reformat □*SM* as appropriate.

The MouseDown event can be used to bring up a pop-up menu. Note however that mouse events are not reported over □*SM* fields because □*SR* uses these to position the cursor.

The illustration shown below was produced as follows :

```
'TEST'      □WC 'Form' 'SM Object' (60 10)(40 50)
'TEST.MB'   □WC 'MenuBar'
'TEST.MB.F' □WC 'Menu' '&File'
'TEST.MB.O' □WC 'Menu' '&Options'
'TEST.B1'   □WC 'Button' '&OK' (84 2)
'TEST.B2'   □WC 'Button' '&Cancel' (84 78)

'TEST.S'    □WC 'SM' (2 2)(80 96)('BCol' 192 192 192)

□SM←↑('The answer is' 5 10)(42 5 30)
```
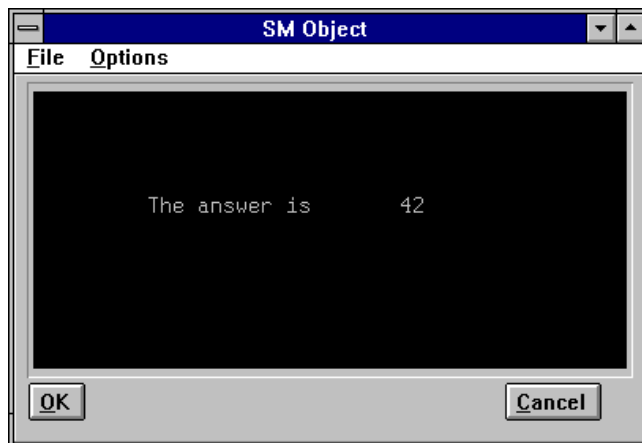
SocketNumber Property

Applies to TCPSocket

The SocketNumber property is an integer whose value is the handle of the socket attached to the TCPSocket object and is generally a read-only property.

The only time that SocketNumber may be specified is when a server replicates (clones) a listening socket to which a client has just connected

SocketType Property

Applies to TCPSocket

The SocketType property is a character vector that specifies the type of the TCP/IP socket. This is either *Stream* (which is the default), or *UDP*.

SocketType must be defined when the object is created and may not be set or changed using *□WS*.

For two Dyalog APL applications to communicate, their TCPSocket objects must have the same SocketType.

SortItems Property

Applies to List

The SortItems property specifies whether or not the items in a List object are sorted. It is Boolean with a default value of 0. If SortItems is 1, the items are automatically sorted in alphabetical order and the object provides word recognition capabilities for selecting an item from the keyboard.

Note that the value of the Items property reflects the order of the items displayed in the List object rather than the order in the array that was used to assign it.

This property may only be initialised when the object is created and cannot subsequently be changed.

Spin   Event 420

Applies to   Spinner, UpDown

If enabled, this event is generated when the user clicks one of the spin buttons in a Spinner object. The event is reported *before* the value of the Thumb property has been updated. You may disable the operation of the spin buttons by disabling this event. You may selectively prevent the user selecting a particular value by returning 0 from a callback function. You may also return a modified event message as a result in order to set the Thumb property to a different value.

The event message reported as the result of *⎕DQ*, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	*'Spin'* or 420
[3] Thumb value:	Integer. The new value of the Thumb property resulting from the user pressing one of the spin buttons.
[4] Adjustment:	Integer. The amount by which the new value of the Thumb differs from its previous value.

Spinner   Object

Purpose	The Spinner object allows the user to enter a value, using an UpDown object to adjust it as required.
Parents	ActiveXControl, CoolBand, Form, Grid, Group, PropertyPage, SubForm, ToolBar, ToolControl
Children	Circle, Ellipse, Font, Marker, Poly, Rect, Text, Timer
Properties	Type, Items, Text, Posn, Size, Coord, Align, Border, Justify, Active, Visible, Event, Thumb, Step, Wrap, Limits, Sizeable, Dragable, FontObj, FCol, BCol, CursorObj, AutoConf, Data, Attach, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, FieldType, MaxLength, Decimals, Password, ValidIfEmpty, ReadOnly, FormatString, Changed, Value, Translate, Accelerator, AcceptFiles, KeepOnClose, Redraw, MethodList, ChildList, EventList, PropList
Events	BadValue, Change, Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, FontCancel, FontOK, GotFocus, Help, KeyError, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Select, SetSpinnerText, Spin
Methods	Animate, ChooseFont, Detach, GetFocus, GetTextSize, ShowSIP

The Spinner object is a composite object that consists of an edit field and a pair of spin buttons. The user may enter a value by typing in the edit field and may adjust the value with the spin buttons. The Spinner may cycle through a pre-defined set of values specified by the Items property or through a range of values specified by the Limits property. The FieldType property supports all of the standard data types, i.e. Char, Numeric, LongNumeric, Currency, Date, LongDate and Time.

The Limits property is a 2-element numeric vector that specifies the minimum and maximum value of the object. The Step property specifies the amount by which the value is incremented or decremented by the spin buttons. The current value in the object is defined by the Thumb and Value properties, which are usually identical. If ReadOnly is 0, the user may type a value into the edit field which will be validated and converted according to the FieldType. In this case, the Value and the Thumb properties may be different.

An *alternative* way to use the Spinner object is to specify the Items property. This defines a discrete set of values through which the user may cycle, and the object behaves rather like a Combo without a drop-down list. In this case, the Limits property is automatically set to (1 , ⊃ρ *Items*), Thumb refers to the *index* into the list of Items, and Step specifies the amount by which this index is updated by the spin buttons. For example, if you set Step to 3, the spin buttons would display every third item.

The Items property may be a character matrix, a vector of character vectors, or a numeric vector and will be formatted according to the FieldType. For example, if you wanted the user to select one of a set of specific dates, you would set the FieldType to Date or LongDate and the Items property to the day numbers (since 1 January 1900) corresponding to the dates you require. The ReadOnly property specifies whether or not the user may enter data into the edit field. A value typed in by the user will be converted and formatted according to the FieldType but need not correspond to a value in Items.

In operation, the value in the Spinner is adjusted by the Step each time one of the spin buttons is clicked. If the user holds a spin button down, the value is adjusted at the rate defined for the keyboard repeat rate. Furthermore, the size of each adjustment is increased according to the length of time the button stays depressed. After 1 second, the amount is increased to (2 × *Step*) after 2 seconds, to (4 × *Step*), after 3 seconds to (8 × *Step*) and so forth until the amount of adjustment exceeds one quarter of the range (*Limits* [2] - *Limits* [1]).

When the value in the spinner reaches its top or bottom limit, it will wrap around to the opposite limit if the value of the Wrap property is 1 (the default). Otherwise it will stick.

The MaxLength property defines the maximum number of characters that the user may type into the edit field. The Decimals property specifies the number of decimal places to which a numeric value is displayed and applies only if the FieldType is Numeric or LongNumeric.

The Spinner generates two special events, Spin and SetSpinnerText. The Spin event is generated each time the value of the Thumb is about to be updated and reports the new value and the difference between it and the current value. You may prevent the Thumb from being updated by returning 0 from a callback function, or you may alter the new value of the Thumb by returning a modified message. The SetSpinnerText event is generated after the Thumb has been reset but *before* the edit field has been updated. It reports the new value of the Thumb and the text that is about to be written into the edit field. By returning a modified event message from a callback, this event allows your application to respond *dynamically* to the spin buttons and to control the text in the edit field directly.

Like an Edit object, the Spinner has a Changed property and generates a Change event when loses the focus after the value of its Text and/or Thumb property has been altered.

If FieldType is Numeric, LongNumeric, Currency, Date, LongDate or Time, the Spinner will generate a BadValue event when it loses the focus if the text in the edit field (i.e. the Text property) is in conflict with the FieldType property and cannot be converted to an appropriate number. If the edit field is empty, a BadValue event will be generated if ValidIfEmpty is 0, but not if it is set to 1.

SplitObj1 Property

Applies to   Splitter

The SplitObj1 property specifies the name of an object managed by a Splitter object. The object must be one of the following types:

Animation	Button	Calendar	Combo
ComboEx	DateTimePicker	Edit	Grid
Group	Label	List	ListView
MDIClient	ProgressBar	RichEdit	Scroll
Spinner	Static	StatusBar	SubForm
TabBar	TabControl	ToolBar	TrackBar
TreeView	UpDown		

If the Style property of the Splitter is '*Vert*', the object specified by SplitObj1 is positioned at (0 0) and sized to occupy the space in its parent to the left of the Splitter, with the Splitter itself attached to its right edge.

If the Style property of the Splitter is '*Horz*', the object specified by SplitObj1 is positioned at (0 0) and sized to occupy the space in its parent above the Splitter, with the Splitter itself attached to its bottom edge.

If SplitObj1 is empty, the Splitter manages the single object named by SplitObj2 and the space to the left or above the Splitter is empty or controlled by another Splitter.

SplitObj2   Property

Applies to   Splitter

The SplitObj2 property specifies the name of one of up to two objects managed by a Splitter object. The object must be one of the following types:

Animation	Button	Calendar	Combo
ComboEx	DateTimePicker	Edit	Grid
Group	Label	List	ListView
MDIClient	ProgressBar	RichEdit	Scroll
Spinner	Static	StatusBar	SubForm
TabBar	TabControl	ToolBar	TrackBar
TreeView	UpDown		

If the Style property of the Splitter is *'Vert'*, the object specified by SplitObj2 is initially positioned at (0 x), where x is half the width of the parent plus the Size of the Splitter, and sized to occupy the space in its parent to the right of the Splitter, with the Splitter itself attached to its left edge.

If the Style property of the Splitter is *'Horz'*, the object specified by SplitObj2 is initially positioned at (y 0), where y is half the height of the parent plus half the Size of the Splitter, and sized to occupy the space in its parent below the Splitter, with the Splitter itself attached to its top edge.

If SplitObj2 is empty, the Splitter manages the single object named by SplitObj1 and the space to the right or below the Splitter is empty or controlled by a second Splitter.

Splitter   Object

Purpose	The Splitter object divides a container into resizable panes.
Parents	ActiveXControl, Form, Group, PropertyPage, SubForm
Children	Timer
Properties	Type, SplitObj1, SplitObj2, Posn, Size, Style, Coord, Align, Active, Visible, Event, BCol, CursorObj, Data, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, EndSplit, Splitting, StartSplit
Methods	Detach

The Splitter divides the client area of a Form or SubForm into resizable panes. Each pane created this way may be empty or be occupied by a single object. If the object in a pane is itself a container object, such as a SubForm, it may have a number of other controls within it.

A single Splitter may manage the geometry of 0, 1 or 2 other objects, which, together with the Splitter itself, share the same parent. The two objects are named by the SplitObj1 and SplitObj2 properties respectively.

A Splitter may manage objects of the following types:

Animation	Button	Calendar	Combo
ComboEx	DateTimePicker	Edit	Grid
Group	Label	List	ListView
MDIClient	ProgressBar	RichEdit	Scroll
Spinner	Static	StatusBar	SubForm
TabBar	TabControl	ToolBar	TrackBar
TreeView	UpDown		

If Style is `'Vert'` (the default), the Splitter is drawn vertically in its parent with the first object (SplitObj1) positioned to its left, and the second object (SplitObj2) to its right as illustrated by the following example.

```
'F'⎕WC'Form' 'Vertical Splitter'('Size' 25 25)
'F.E1'⎕WC'Edit'(10 6ρ'Edit 1')('Style' 'Multi')
'F.E2'⎕WC'Edit'(10 6ρ'Edit 2')('Style' 'Multi')
'F.S'⎕WC'Splitter' 'F.E1' 'F.E2'
```
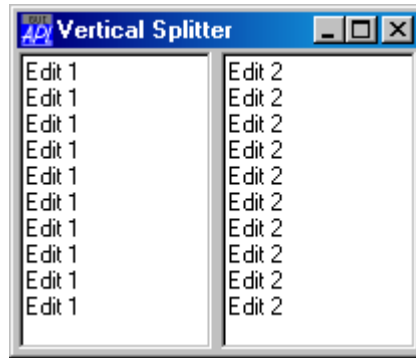

If Style is `'Horz'`, the Splitter is drawn horizontally in its parent with the first object (SplitObj1) positioned above, and the second object (SplitObj2) below.
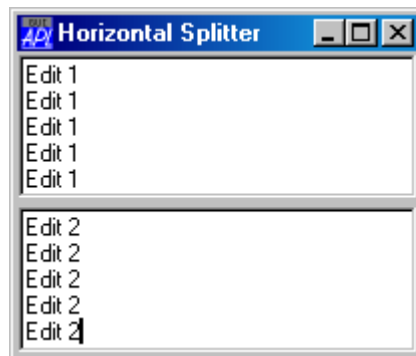
The Style property must be set when the object is created with ⎕WC and may not subsequently be changed using ⎕WS.

The Posn and Size properties are partially read-only, in that only one dimension of the value may be specified. If Style is `'Vert'`, you may specify the x-coordinate and the width of the Splitter, but you may not specify its y-coordinate nor its height. If Style is `'Horz'`, you may specify the y-coordinate and the width of the Splitter, but you may not specify its x-coordinate nor its length.

When the user positions the mouse pointer directly over the Splitter object, the cursor changes (by default) to a double-headed arrow (direction in accordance with Style). The user may now depress the left mouse button and drag the Splitter to a new position, resizing the objects named by SplitObj1 and SplitObj2 in the process.

You can select a different cursor using the CursorObj property. Note that setting the CursorObj property to 0 selects the default cursor, which is the appropriate double-headed arrow.

When the user depresses the mouse button, the Splitter generates a StartSplit event. When the user releases the mouse button, the Splitter generates an EndSplit event. If full-drag is in effect, the Splitter also reports Splitting events as it is dragged. All these events report the new or current position of the Splitter object and are provided for information only.

Note that the objects named by SplitObj1 and SplitObj2 and any sub-objects they contain will generate Configure events when they are resized by the Splitter.

Alignment

The Align property specifies how a Splitter behaves when its parent is resized and may be `'None'`, `'Left'`, `'Right'`, `'Top'` or `'Bottom'`.

If Align is `'None'`, the Splitter moves as its parent is resized, so that it divides its parent in the same *proportions* as before. This is the default.

Any other value of Align attaches the Splitter to the corresponding edge of its parent. For example, if Align is `'Left'`, the width of the object to the *left* of the Splitter remains fixed when its parent is resized horizontally by the user.

Like the Style property, Align may be set only when the object is created with `⎕WC` and may not subsequently be changed using `⎕WS`.
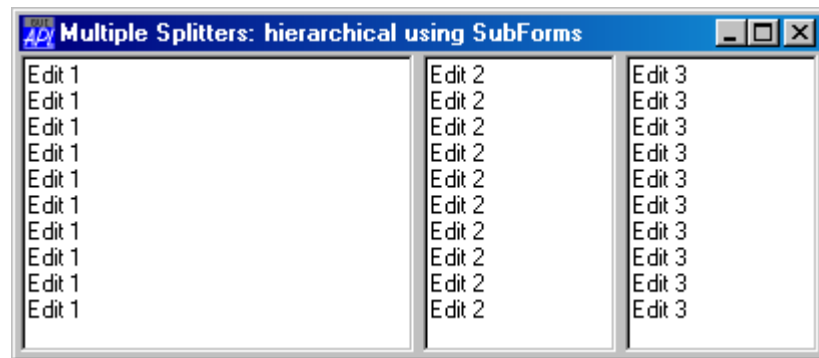
Using Multiple Splitters

If you want to divide a Form into more than 2 resizable panes, there are two possible approaches, each with its own different characteristics.

The first approach, illustrated below, is a hierarchical one using SubForms. This example shows how you can create a Form containing three resizable Edit objects.

```
TITLE← 'Multiple Splitters: hierarchical using SubForms'
'F'⎕WC'Form' TITLE ('Size' 25 50)
'F.E1'⎕WC'Edit'(10 6ρ'Edit 1')('Style' 'Multi')
'F.SF1'⎕WC'SubForm'('EdgeStyle' 'Default')
'F.S1'⎕WC'Splitter' 'F.E1' 'F.SF1'
'F.SF1.E1'⎕WC'Edit'(10 6ρ'Edit 2')('Style' 'Multi')
'F.SF1.E2'⎕WC'Edit'(10 6ρ'Edit 3')('Style' 'Multi')
'F.SF1.S1'⎕WC'Splitter' 'F.SF1.E1' 'F.SF1.E2'
```

First, you create an Edit, a SubForm, and a Splitter as children of the *Form*, using the Splitter to divide the Form into two panes, one for the Edit and the other for the SubForm. Next, you create two Edit objects and a Splitter as children of the *SubForm*, using the second Splitter to divide the SubForm into two. You can continue with this approach to any reasonable depth.

Notice that when the first Splitter is shifted to the left, both panes in the SubForm expand equally (because Align is `'None'`) as shown below.
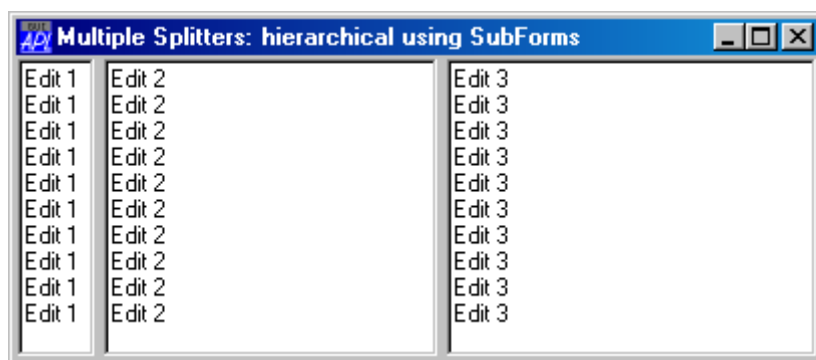
The second approach, illustrated by the following example, is to create multiple Splitters *at the same level*, i.e. owned by the same parent. In this case, the third Edit object `F.E3` is unaffected by movement of the leftmost Splitter `F.S1`.

```
'F'⎕WC'Form' 'Multiple Splitters: non-hierarchical'
                                ('Size' 25 50)
'F.E1'⎕WC'Edit'(10 6ρ'Edit 1')('Style' 'Multi')
'F.E2'⎕WC'Edit'(10 6ρ'Edit 2')('Style' 'Multi')
'F.E3'⎕WC'Edit'(10 6ρ'Edit 3')('Style' 'Multi')
'F.S1'⎕WC'Splitter' 'F.E1'
'F.S2'⎕WC'Splitter' 'F.E2' 'F.E3'
```
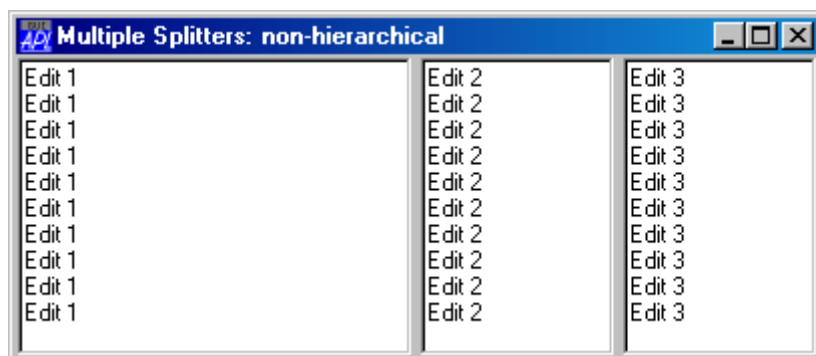
Using the non-hierarchical approach, horizontal and vertical Splitters may be combined in interesting ways as illustrated below. This can also be achieved using nested SubForms, but at the expense of a complex object hierarchy.

Notice that in this example, with the exception of the last Splitter *F.S4*, it is necessary only to specify the SplitObj1 property for each of the Splitters. The reason is that the first four Splitters only manage one object *directly*. For example, the object to the right of *F.S1* is in fact a horizontal Splitter *F.S2*. Dragging *F.S1* changes the length of *F.S2* which in turn changes the width of *F.E2*. and *F.E3*.

```
'F'⎕WC'Form' 'Combining Horizontal and Vertical Splitters'
'F.E1'⎕WC'Edit'(20 6ρ'Edit 1')('Style' 'Multi')
'F.E2'⎕WC'Edit'(10 6ρ'Edit 2')('Style' 'Multi')
'F.E3'⎕WC'Edit'(10 6ρ'Edit 3')('Style' 'Multi')
'F.E4'⎕WC'Edit'(5 6ρ'Edit 4')('Style' 'Multi')
'F.E5'⎕WC'Edit'(5 6ρ'Edit 5')('Style' 'Multi')

'F.S1'⎕WC'Splitter' 'F.E1'('Style' 'Vert')
'F.S2'⎕WC'Splitter' 'F.E2'('Style' 'Horz')
'F.S3'⎕WC'Splitter' 'F.E3'('Style' 'Vert')
'F.S4'⎕WC'Splitter' 'F.E4' 'F.E5'('Style' 'Horz')
```
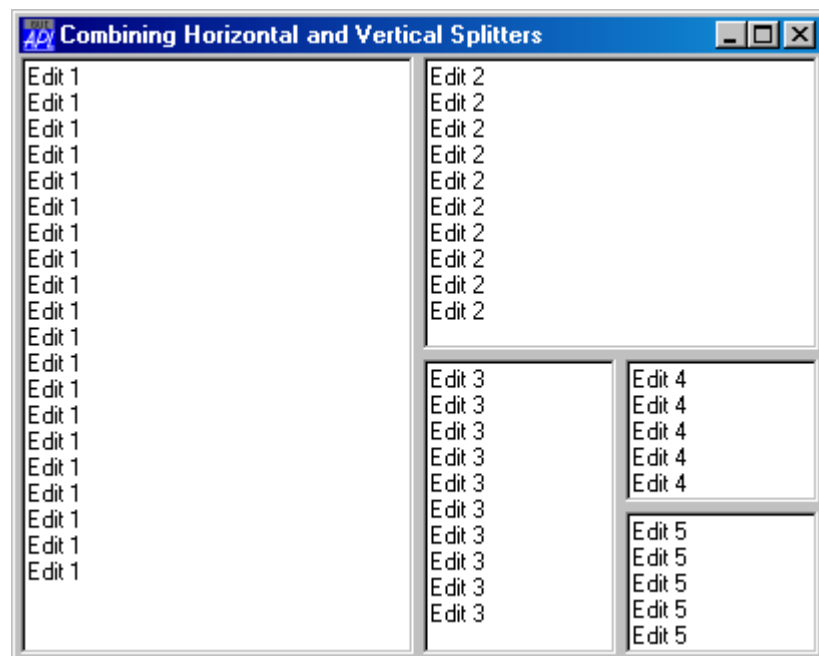
Colliding Splitters

If you have two or more vertical Splitters or two or more horizontal Splitters in the same parent object, it is possible for the user to make the Splitters *collide*. This can occur by dragging one of the Splitters into the other, or, unless both Splitters have Align set to `'None'`, by shrinking the parent.

When Splitters collide, the object being dragged by the user (a Splitter or a border of the parent) takes precedence over the setting of Align, and temporarily *pushes* other Splitters along in its direction of travel. If and when the operation is reversed, the other Splitters are *pulled* back to their original positions.

Splitting   Event 281

Applies to   Splitter

If enabled, this event is reported while a Splitter object is being dragged, between a StartSplit and an EndSplit. This event is only reported if full-drag is enabled.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to ¯1 or by returning 0 from a callback function.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 6-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	`'Splitting'` or 281
[3] Y:	y-position of top left corner
[4] X:	x-position of top left corner
[5] H:	height of the Splitter
[6] W:	width of the Splitter

See also StartSplit, EndSplit.

Start Property

Applies to Circle, Ellipse

This property specifies one or more start-angles for an arc, pie-slice, or chord of a circle or ellipse. It may be used in conjunction with End which specifies end angles. Angles are measured counter-clockwise from the x-axis at the centre of the object.

If a single arc is being drawn, Start is a single number that specifies the start angle of the arc in radians (`0 -> ○2`). If multiple arcs are being drawn, Start is either a single number as before (the start angle for several concentric arcs) or a numeric vector with one element per arc.

If End is not specified, the default value of Start is 0. Otherwise, the default value of Start is (`0,¯1↓+\End`).

StartIn Property

Applies to BrowseBox

The StartIn property is a character string that specifies the start point and root for a BrowseBox object.

Only the specified folder and its subfolders appear in the dialog box. The user cannot browse higher in the folder architecture than this folder.

The default value for StartIn is an empty vector which means that the root of the browse dialog is the desktop.

StartSplit   Event 280

Applies to   Splitter

If enabled, this event is reported when the user depresses the left mouse button over a Splitter object to signify the beginning of a drag operation.

This event is reported for information alone. You may not disable or nullify the event by setting the action code for the event to ¯1 or by returning 0 from a callback function.

The event message reported as the result of `⎕DQ`, or supplied as the right argument to your callback function, is a 2-element vector as follows :

     [1] Object: ref or character vector

     [2] Event name or code: *'StartSplit'* or 280

See also EndSplit, Splitting.

State   Property

Applies to   Button, Form, SubForm, TabButton, ToolButton

This property determines the state of a Button, TabButton, ToolButton, Form, or SubForm. It is a single number with the value 0 (the default), 1, or 2 (Form and SubForm).

If the Style property is *'Push'*, a State of 0 means that the pushbutton is displayed normally (out). If its State is 1, the pushbutton is displayed depressed (in).

If the Style property is *'Radio'* or *'Check'*, 0 means "not selected" and 1 means "selected". Note that only one of a group of buttons with Style *'Radio'* that share the same parent may have State 1. Setting State to 1 automatically deselects all the others in the group.

For a Form or SubForm, a value of State of 0 means that the Form is currently displayed in its "normal" state. 1 means that the Form is currently minimised (displayed as an icon). The value 2 indicates that the Form is maximised and displayed full-screen. The State of a Form can be changed using `⎕WS`.

StateChange   Event 35

Applies to   Form, SubForm

This event is generated by a Form or SubForm when the user attempts to change the
State of a Form, by minimising it, maximising it, or restoring it from a minimised or
maximised state. The event is reported **before** the window changes state. You may
prevent the state change by disabling the event (action code ¯1) or by returning a 0
result from an attached callback function.

The event message reported as the result of ⎕DQ, or supplied as the right argument to
your callback function, is a 3-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	*'StateChange'* or 35
[3] Window state:	0 (about to be restored)
	1 (about to be minimised)
	2 (about to be maximised)

Static   Object

Purpose	This object is primarily used to display graphics in a sub-window.
Parents	ActiveXControl, CoolBand, Form, Group, PropertyPage, SubForm, ToolBar, ToolControl
Children	Circle, Cursor, Ellipse, Font, Image, Locator, Marker, Metafile, Poly, Rect, Text, Timer
Properties	Type, Posn, Size, Style, Coord, Border, Active, Visible, Event, Sizeable, Dragable, FontObj, FCol, BCol, Picture, CursorObj, AutoConf, YRange, XRange, Data, Attach, TextSize, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, Translate, Accelerator, AcceptFiles, KeepOnClose, Redraw, MethodList, ChildList, EventList, PropList
Events	Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, FontCancel, FontOK, Help, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Select
Methods	Animate, ChooseFont, Detach, GetFocus, GetTextSize, ShowSIP

The overall appearance of an empty Static object is controlled by the value of its Style property which may be one of the following character vectors :

```
'BlackFrame'    'BlackBox'
'GreyFrame'     'GreyBox'
'GrayFrame'     'GrayBox'
'WhiteFrame'    'WhiteBox'
```

Note that the colours implied by the Style are not "hard-coded" but are actually defined by the current Windows colour scheme as follows :

Black	Window Border Colour
Grey/Gray	Desktop Colour
White	Window Background Colour

If the background colour of the Form is also set to the Window Background Colour, it follows that the Styles `'WhiteFrame'` and `'WhiteBox'` make the Static itself invisible (against the background), although the **contents** of the Static will show. This makes the Static appear like an invisible clipping window.

StatusBar   Object

Purpose	This object is used to manage StatusField objects which display information for the user.
Parents	ActiveXControl, CoolBand, Form, SubForm
Children	Bitmap, BrowseBox, Circle, Cursor, Ellipse, FileBox, Font, Icon, Image, Marker, Poly, ProgressBar, Rect, StatusField, Text, Timer
Properties	Type, Posn, Size, Coord, Align, Border, Active, Visible, Event, VScroll, HScroll, Sizeable, FontObj, FCol, BCol, Picture, CursorObj, AutoConf, YRange, XRange, Data, Attach, TextSize, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, Translate, Accelerator, AcceptFiles, KeepOnClose, Redraw, MethodList, ChildList, EventList, PropList
Events	Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, FontCancel, FontOK, Help, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Select
Methods	Animate, ChooseFont, Detach, GetFocus, GetTextSize, ShowSIP

The StatusBar is a container object that manages StatusFields. StatusField objects display textual information and are typically used for help messages and for monitoring the status of an application. They can also be used to automatically report the status of the Caps Lock, Num Lock , Scroll Lock, and Insert keys. The picture below illustrates a StatusBar containing 3 StatusFields which was produced by the following statements:.

```
'TEST'⎕WC'Form' 'The StatusBar Object'
              ('EdgeStyle' 'Default')
'TEST.SB'⎕WC'StatusBar'
'TEST.SB.S1'⎕WC'StatusField' 'Field1:' 'text1'
'TEST.SB.S2'⎕WC'StatusField' 'Field2:' 'text2'
'TEST.SB.S3'⎕WC'StatusField' 'Field3:' 'text3'
```
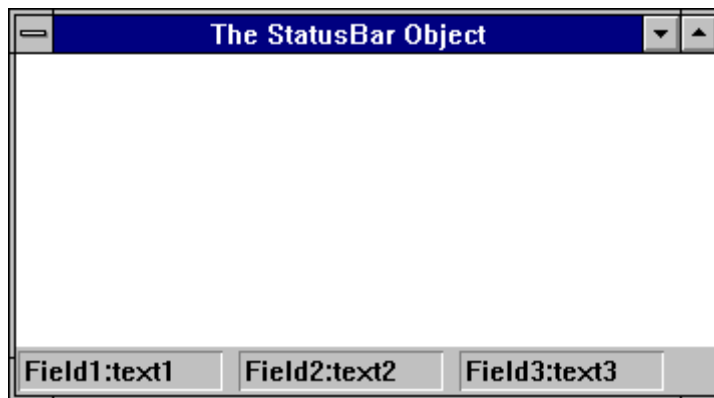
The Align property determines to which side of the parent Form or SubForm the StatusBar is attached. By default, a StatusBar is positioned along the lower edge of the Form (Align *'Bottom'*) and is 24 pixels high. Using the Align, Posn and Size properties you may create StatusBars in different positions and with differing sizes if you wish. Notice that the Align property controls how the StatusBar reacts to its parent Form being resized. If Align is *'Top'* or *Bottom*, the StatusBar remains fixed in height but stretches and shrinks sideways with the Form. If Align is *'Left'* or *'Right'*, the StatusBar remains fixed in width and stretches and shrinks vertically with the Form.

By default a StatusBar has a Button Face colour background and the value of its EdgeStyle property is *'Default'*. This gives it the appearance shown above.

Unless you specify the position and size of its children, a StatusBar automatically manages their geometry. The first StatusField is positioned just inside its top left corner. If Align is *'Top'* or *'Bottom'*, the next StatusField is positioned alongside the first but with a small gap between them. Subsequent StatusFields are added in a similar fashion. If Align is *'Left'* or *'Right'*, the second and subsequent StatusFields are added below the first with a similar gap between them. In either case you can position and size the StatusFields explicitly if you wish.

If you attempt to add a StatusField that would extend beyond the right edge (Align *'Top'* or *'Bottom'*) or bottom edge (Align *'Left'* or *'Right'*) the behaviour depends upon the value of HScroll or VScroll. If HScroll is 0 (the default) and Align is *'Top'* or *'Bottom'*, the StatusField is added below the first one, thereby starting a new row. If VScroll is 0 (the default) and Align is *'Left'* or *'Right'*, it is added to the right of the first one thereby starting a new column. If HScroll or VScroll is ¯1 or ¯2, the new StatusField is simply positioned in the same row or column and may be scrolled into view using a mini scrollbar. A value for HScroll or VScroll of ¯1 causes the mini scrollbar to be permanently present in the ScrollBar. A value of ¯2 causes it to appear only when required.

StatusField   Object

Purpose	This object is used to display information for the user.
Parents	StatusBar
Children	Menu, Timer
Properties	Type, Caption, Text, Posn, Size, Style, Coord, Align, Border, Visible, Event, Sizeable, Dragable, FontObj, FCol, BCol, Picture, AutoConf, Data, Attach, EdgeStyle, Translate, Accelerator, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, DropObjects, MouseDblClick, MouseDown, MouseMove, MouseUp, Select
Methods	Detach

The StatusField object provides an area for displaying context sensitive help messages, keyboard status, and other application dependant information.

By default a StatusField is a recessed rectangle in which information is displayed. It has a Caption and a Text property, which by default are empty, but either or both of which can be used to present information. The Caption is left justified in the field and the Text is displayed immediately to its right. Typically, you would use the Caption property as a title to describe the information that the StatusField displays, and the Text property to show its current value. However, you are not obliged to use both of them and you can achieve most effects with just one.

Note that when the StatusField is used to display hints, it is its Text property that is used.

A StatusField may be used to monitor the status of the keyboard and this is controlled by its Style property. The default value for Style is an empty vector. However, you can set it to monitor various keyboard states as follows :

Style	Description
CapsLock	Monitors state of Caps Lock key
ScrollLock	Monitors state of Scroll Lock key
NumLock	Monitors state of Num Lock key
KeyMode	Monitors the keyboard mode (APL/ASCII)
InsRep	Monitors the state of the Insert key

In each case, the Text property of the StatusField is used to display the keyboard status. If Style is CapsLock, ScrollLock or NumLock, the field displays "Caps", "Num" or "Scroll" if the corresponding mode is selected and is blank if not.

If Style is InsRep, the StatusField displays either "Ins" or "Rep". Initially it always displays "Ins" and then toggles between "Rep" and "Ins" each time the Insert key is pressed.

If Style is KeyMode, the StatusField displays the name for the current keyboard mode which is defined in the input table being used. For the 2-mode tables APL_US.DIN, APL_UK.DIN etc., the mode name displayed is either "Apl" or "Asc". The unified tables have no modes so a StatusField with this Style does nothing.

If Style is set to one of the above, you may still use the Caption property to give the StatusField a title. You may even set the value of the Text property, but be aware that this value will be reset when the user next presses the key the StatusField is monitoring.

Step   Property

Applies to   Form, Locator, ProgressBar, Scroll, Spinner, SubForm, TrackBar, UpDown

For a Form, Scroll and SubForm, this property determines the size of changes reported when the user clicks a scroll arrow (small change) or clicks on the body of the scrollbar (large change). The object's Thumb property increases or decreases by this amount.

For a Scroll object, Step is a 2-element numeric vector whose first element specifies the value of the "small change" and whose second element specifies the value of the "large change".

For a Form or SubForm, Step is a 4-element numeric vector. The first two elements refer to the Form's vertical scrollbar and the second two elements refer to the Form's horizontal scrollbar.

For these objects, values of Step must be between 1 and the value of the Range property.

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

Style   Property

Applies to   Button, Calendar, Combo, ComboEx, DateTimePicker, Edit, FileBox, Icon, List, ListView, Locator, Marker, MenuItem, MsgBox, ProgressBar, PropertySheet, Separator, Splitter, Static, StatusField, TabControl, TCPSocket, ToolButton, ToolControl, TrackBar

This property determines a particular style of object within the general category of Type. It is a character vector whose value depends upon the type of object.

For a Button, Style may be *'Push'*, *'Radio'* or *'Check'*.

'Push' specifies that the button appears and behaves like a pushbutton (sometimes also called a command button).

'Radio' means that the button is displayed as a small circle accompanied by a description. When the button is selected, the circle is filled in. In a group of buttons with Style *'Radio'* that share the same parent, only one of them may be selected. This style of button is generally known as a "radio-button" or an "option button".

'Check' means that the button is displayed as a small box accompanied by a description. When the button is selected a cross appears in the box. This style of button is known as a "check-box".

For a Calendar object, The Style property may be either *'Single'* (the default) or *'Multi'*. If Style is *'Single'*, the user may select a single date. If Style is *'Multi'*, the user may select a contiguous range of dates.

For a Combo or ComboEx object, Style may be *'Simple'* *'DropEdit'* or *'Drop'* (the default). *'Simple'* specifies a simple combo box in which the associated list box is displayed at all times. The other two styles provide list boxes which "drop down" when the user clicks on a symbol displayed to the right of the Combo's edit field. A *'DropEdit'* Style allows the user to type (anything) in the edit field. A *'Drop'* Style does not and forces the contents of the edit field to be either empty or one of the choices specified by Items.

For a DateTimePicker, Style may be either *'Combo'* (the default) or *'UpDown'*.

For an Edit object, Style may be *'Single'* or *'Multi'*. If Style is *'Single'* the object displays only a single line of text and the user may not enter any more lines. If the Style is *'Multi'* the number of lines displayed is governed by the Rows or Size property and the user may insert, add or delete lines as desired.

For FileBox, List and ListView objects, Style may be `'Single'` or `'Multi'`. If the Style is `'Single'` only one file or item can be selected. If Style is `'Multi'`, several files or items can be selected.

For an Icon, Style may be `'Large'` (the default) or `'Small'` and specifies the size of the icon (32x32 or 16x16) to be loaded from a file.

For a Locator, Style may be `'Point'`, `'Rect'` (the default), `'Line'` or `'Ellipse'`. It specifies the shape that is drawn as the user moves the mouse.

For a MsgBox, the Style property determines the type of icon which is displayed in it. This is a character vector with one of the following values :

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

For a Splitter, the Style property specifies the orientation of the Splitter and may be `'Vert'` (the default) or `'Horz'`.

For a Static object, Style defines its appearance, and may be one of :

`'BlackFrame'`	`'BlackBox'`
`'GreyFrame'`	`'GreyBox'`
`'GrayFrame'`	`'GrayBox'`
`'WhiteFrame'`	`'WhiteBox'`

A StatusField may be used to monitor the state of a key on the keyboard. If so, its Style property determines the key it monitors and may be one of the following:

`'CapsLock'`	Monitors state of Caps Lock key
`'ScrollLock'`	Monitors state of Scroll Lock key
`'NumLock'`	Monitors state of Num Lock key
`'KeyMode'`	Monitors the keyboard mode (APL/ASCII)
`'InsRep'`	Monitors the state of the Insert key

For a TabControl, the Style property determines the appearance of its TabButton children, and may be `'Tabs'` (the default), `'Buttons'` or `'FlatButtons'`.

For a TCPSocket, the Style property is a character vector that specifies the type of data transmitted or received by the socket; it may be `'Char'`, `'Raw'`, or `'APL'`.

For a ToolButton, the Style property specifies the behaviour of the button and may be `'Push'` (the default), `'Check'`, `'Radio'`, `'DropDown'` or `'Separator'`.

For a ToolControl, the Style property determines the appearance of its ToolButton children and may be `'Buttons'`, `'FlatButtons'` (the default), `'List'` or `'FlatList'`.

For a TrackBar, the Style property determines the appearance and behaviour of the TrackBar and may be `'Standard'` (the default) or `'Selection'`.

SubForm   Object

Purpose	This object represents a window that is owned by and constrained within another Form or an MDIClient.
Parents	ActiveXControl, CoolBand, Form, Group, MDIClient, PropertyPage, SubForm, TabControl, ToolBar, ToolControl
Children	Animation, Bitmap, BrowseBox, Button, Calendar, Circle, ColorButton, Combo, ComboEx, Cursor, DateTimePicker, Edit, Ellipse, FileBox, Font, Grid, Group, Icon, Image, ImageList, Label, List, ListView, Locator, Marker, MDIClient, Menu, MenuBar, Metafile, MsgBox, OCXClass, Poly, ProgressBar, PropertySheet, Rect, RichEdit, Scroll, SM, Spinner, Splitter, Static, StatusBar, SubForm, TabBar, TabControl, TCPSocket, Text, Timer, TipField, ToolBar, ToolControl, TrackBar, TreeView, UpDown
Properties	Type, Caption, Posn, Size, Coord, State, Border, Active, Visible, Event, Thumb, Range, Step, VScroll, HScroll, Sizeable, Moveable, SysMenu, MaxButton, MinButton, HelpButton, FontObj, BCol, Picture, OnTop, IconObj, CursorObj, AutoConf, YRange, XRange, Data, Attach, TextSize, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, TabObj, Translate, Accelerator, AcceptFiles, KeepOnClose, Dockable, Docked, DockShowCaption, DockChildren, UndocksToRoot, Redraw, MethodList, ChildList, EventList, PropList
Events	Close, Configure, ContextMenu, Create, DockAccept, DockCancel, DockEnd, DockMove, DockRequest, DockStart, DragDrop, DropFiles, DropObjects, Expose, FontCancel, FontOK, FrameContextMenu, GotFocus, Help, HScroll, KeyPress, LostFocus, MDIActivate, MDIDeactivate, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Select, StateChange, VScroll
Methods	Animate, ChooseFont, Detach, GetFocus, GetTextSize, ShowSIP

If the SubForm is the child of a Form, it is by default a simple featureless window that occupies the entire client area (excluding standard ToolBars, StatusBars and TabBars) of its parent. The properties that control its appearance, including Sizeable, Moveable, SysMenu, Border, MaxButton and MinButton, all default to 0. The EdgeStyle property also defaults to `'None'`, so the background of the SubForm defaults to the Window Background colour.

If the SubForm is the child of an MDIClient, its default appearance is the same as for a top-level Form. By default its size is 25% of its parent client area and it is positioned in the centre of its parent object.

The Posn property specifies the location of the **internal** top-left corner of the SubForm relative to its parent. If the SubForm has a title bar, border, or a 3-dimensional shadow, you must allow sufficient space for these components. Similarly, the Size property specifies the internal size of the SubForm excluding the title bar and border.

A SubForm is constrained so that it cannot be moved outside its parent. In all other respects it behaves in a similar manner to a Form object. See Form object and the descriptions of its properties for further details.

SysColorChange   Event 134

Applies to   Root

If enabled, this event is reported when the user or another application updates the system colour palette. The event is reported after the change has taken place and cannot be disabled or inhibited in any way. If you want your application to respond to colour palette changes, this event gives you the opportunity of doing so.

The event message reported as the result of ⎕DQ, or supplied as the right argument to your callback function, is a 2-element vector as follows :

> [1] Object: ref or character vector
>
> [2] Event code: '*SysColorChange*' or 134

SysMenu   Property

Applies to   Form, SubForm

This property determines whether or not a Form or SubForm has a "System Menu" box in the top-left corner of its border. Pressing the left mouse button in this box brings up the standard window control menu for the Form. Double clicking the box closes the Form.

SysMenu is a single number with the value 0 (no System menu box) or 1 (System Menu box is provided). The default is 1.

If any of the SysMenu, MaxButton, MinButton and Moveable properties are set to 1, the Form or SubForm has a title bar.

SysTrayItem   Object

Purpose	The SysTrayItem object represents an item that you can create in the Windows System Tray.
Parents	Form, Root
Children	Icon, Menu, Timer
Properties	Type, Event, IconObj, Data, Tip, Translate, Popup, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, MouseDblClick, MouseDown, MouseMove, MouseUp
Methods	Detach, Wait

The SysTrayItem object appears as an icon in the Windows System Tray and allows the user to interact with your application even if it is minimised or has no other visible presence.

Interaction is provided through a pop-up menu that is displayed when the user clicks on the SysTrayItem. The SysTrayItem does not support mouse or keyboard events directly.

The IconObj property specifies the name of an Icon object used to display the SysTrayItem. If not specified, the default is the standard Dyalog APL icon.

The Popup property specifies the name of a Menu object (which may be a child of the SysTrayItem). The Menu object is displayed automatically when the user clicks on the SysTrayItem icon. The Menu should contain one or more MenuItem objects with suitable callback functions attached.

Unlike other popup menus, the SysTrayItem menu is not activated by an explicit (modal) *⎕DQ* but is posted automatically for you. The MenuItem callbacks will be executed by the current *⎕DQ*, with the exception of modal *⎕DQ*s on MsgBox, FileBox, Locator and other popup Menu objects. For example, if your application is in a modal *⎕DQ* on a Form, that *⎕DQ* will react to and action events on the SysTrayItem menu, even though it is not explicitly included in the list of objects being *⎕DQ*'ed.

The Tip property specifies a character string to be displayed when the user hovers the mouse over the SysTrayItem. This is displayed using the user's current setting for Tip text and it is not possible to change this appearance.

TabBar   Object

Purpose	To manage a set of TabBtn objects.
Parents	ActiveXControl, Form, SubForm
Children	Circle, Ellipse, Font, Marker, Poly, Rect, TabBtn, Text, Timer
Properties	Type, Posn, Size, Coord, Align, Active, Visible, Event, VScroll, HScroll, Sizeable, FontObj, BCol, Picture, OnTop, IconObj, CursorObj, AutoConf, YRange, XRange, Data, Attach, TextSize, Handle, Hint, HintObj, Tip, TipObj, TabObj, Translate, Accelerator, AcceptFiles, KeepOnClose, Redraw, MethodList, ChildList, EventList, PropList
Events	Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, Help, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Select
Methods	Animate, Detach, GetFocus, GetTextSize, ShowSIP

The TabBar object manages a group of TabBtn objects. These are associated with a set of SubForm objects which are positioned on top of one another. When the user clicks on a TabBtn, the corresponding SubForm is brought to the top and given the focus.

TabBar and TabBtn objects were implemented before Windows provided direct support for tabbed dialogs, and have been superceded by TabControl and TabButton objects. Please use these instead.

By default, a TabBar is a flat bar stretched across the bottom of its parent form. You can alter its appearance using its EdgeStyle property and you can control its alignment with its Align property. Align can be set to Top (the default) , Bottom, Left or Right and causes the TabBar to be attached to the corresponding edge of the Form. A TabBar aligned Top or Bottom will automatically stretch or shrink horizontally when its parent Form is resized, but it will remain fixed vertically. A TabBar aligned Left or Right will stretch vertically but will remain fixed horizontally. By default a TabBar occupies the entire width or length of the side of the Form to which it is attached and is 17 pixels high or wide. You can change these defaults using the Posn and Size properties.

The alignment of a TabBar also determines the orientation of its TabBtns. TabBars aligned Top or Bottom cause their TabBtns to be drawn left to right with the free edge of the TabBtns facing upwards or downwards respectively. TabBars aligned Left or Right draw their TabBtns downwards with their free edges facing left or right respectively.

By default, TabBtn objects are positioned along the inner edge of the TabBar. This is the edge closest to the SubForms they will tab. They are also positioned so that they overlap one another horizontally or vertically according to the Align property.

The HScroll and VScroll properties determine what happens when the end of the TabBar is reached. If HScroll or VScroll is 0 (the default) a TabBtn that would otherwise extend beyond the TabBar is instead positioned immediately above, below or alongside the first TabBtn in the TabBar, thereby starting a new row or column. Note however that the TabBar is not automatically resized vertically to accommodate a second row or column. If you want a multi-flight TabBar you have to set its height or width explicitly. If HScroll or VScroll is ⁻1 or ⁻2, TabBtns continue to be added along the TabBar even though they extend beyond its boundary and may be scrolled into view using a mini scrollbar. If HScroll is ⁻1, the scrollbar is shown whether or not any controls extend beyond the TabBar. If HScroll is ⁻2, the scrollbar appears only if required and may appear or disappear when the user resizes the parent Form.

If you specify a value for its Posn property, a TabBtn will be placed at the requested position regardless of the value of Style, HScroll or VScroll. However, the next control added will take its default position from the previous one according to the value of these properties. Thus if you wish to group your controls together with spaces between the groups, you need only specify the position of the first one in each group.

If you specify a value for its Posn property, a TabBtn will be placed at the requested position regardless of the value of Align. However, the next TabBtn added will take its default position from the previous one. Thus if you wish to group your TabBtns together with spaces between the groups, you need only specify the position of the first one in each group.

TabBtn   Object

Purpose	To tab a SubForm.
Parents	TabBar
Children	Timer
Properties	Type, Caption, Posn, Size, Align, Border, Active, Visible, Event, FontObj, FCol, BCol, AutoConf, Data, Attach, EdgeStyle, TabObj, Translate, Accelerator, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, DropObjects, FontCancel, FontOK, MouseDblClick, MouseDown, MouseMove, MouseUp, Select
Methods	ChooseFont, Detach

TabBtn objects are associated with SubForms which are positioned on top of one another. When the user clicks on a TabBtn, the corresponding SubForm is brought to the top and given the focus.

TabBar and TabBtn objects were implemented before Windows provided direct support for tabbed dialogs, and have been superceded by TabControl and TabButton objects. Please use these instead.

The appearance of a TabBtn is determined by its EdgeStyle, Border and Caption properties. These take their defaults from the SubForm with which the TabBtn is associated. Thus there is generally no need to specify them. BCol also defaults to that of its associated SubForm.

The position of a TabBtn is normally determined by its parent TabBar and its default size is fixed (22 x 80 pixels), and not related to the size of its Caption. These defaults can be overridden using the Posn and Size properties.

A SubForm is associated with a TabBtn by setting the TabObj property of the *SubForm* to the name of the TabBtn. The TabObj property of the TabBtn is a read-only property that contains the name of the associated SubForm.

TabButton Object

Purpose	The TabButton object represents an individual tab or button in a TabControl.
Parents	TabControl
Children	Timer
Properties	Type, Caption, Posn, Size, State, Event, ImageIndex, Data, Tip, TabObj, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, Select
Methods	Detach

The TabButton object represents an individual tab or button in a TabControl and **requires the Windows Custom Control Library COMCTL32.DLL, Version 4.71 or higher.**

The position and size of a TabButton object are entirely determined by its parent TabControl and may not be altered. For this reason, the Posn and Size properties are read-only.

The Caption property specifies the text that appears on the button or tab.

A picture is specified by setting the ImageIndex property of the TabButton. This is a number that points to a particular icon or bitmap defined in an ImageList object whose name is specified by the ImageListObj property of the parent TabControl.

Note that all TabButton objects share the same font which is defined by the FontObj property of the TabControl.

The foreground and background colours of the TabButton object are fixed.

When used as a tab, a TabButton is normally attached to a SubForm by the TabObj property of the SubForm. The TabObj property of the TabButton itself is a read-only property that reports the name of the SubForm to which the TabButton is attached.

The State property reports the (selected) state of a TabButton and applies only when its parent TabControl has Style set to *'Buttons'* or *'FlatButtons'* and MultiSelect set to 1.

TabControl   Object

Purpose	The TabControl object provides access to the standard Windows 95/NT tab control.
Parents	ActiveXControl, CoolBand, Form, SubForm
Children	ImageList, SubForm, TabButton, Timer
Properties	Type, Posn, Size, Style, Event, ImageListObj, FontObj, Data, Attach, TabObj, KeepOnClose, MultiLine, TabSize, Justify, TabJustify, Align, MultiSelect, TabFocus, HotTrack, ScrollOpposite, FlatSeparators, MethodList, ChildList, EventList, PropList
Events	Close, Create
Methods	Animate, Detach, GetFocus, GetTextSize, ShowSIP

The TabControl object requires the Windows Custom Control Library COMCTL32.DLL, Version 4.71 or higher.

The standard tab control is analogous to a set of dividers in a notebook and allows you to define a set of *pages* that occupy the same area of a window or dialog box. Each page consists of a set of information or a group of controls that the application displays when the user selects the corresponding tab.

A special type of tab control displays tabs that look like buttons. For example, the Windows 98 TaskBar is such a tab control.

Because the TabControl object uses the standard Windows 95/NT tab control, it is not available in Dyalog APL Version 7. If you need to support Version 7, you should use the TabBar object instead. However, if your application is designed only for Dyalog APL Version 8, the TabControl is generally preferable.

The overall appearance of the TabControl is determined by the Style property which may be *'Tabs'* (the default), *'Buttons'* or *'FlatButtons'*.

Individual tabs or buttons are represented by TabButton objects which should be created as children of the TabControl object. Optional captions and pictures are specified by the Caption and ImageIndex properties of the individual TabButton objects themselves. Otherwise, the appearance of the tabs or buttons is determined by properties of the TabControl itself.

To implement a multiple page tabbed dialog, illustrated below, you should create a Form, then a TabControl with Style `'Tabs'` as a child of the Form. Next, create one or more pairs of TabButton and SubForm objects as children of the TabControl. You associate each SubForm with a particular tab by setting its TabObj property to the name of the associated TabButton object. Making the SubForms children of the TabControl ensures that, by default, they will automatically be resized correctly. You may alternatively create your SubForms as children of the main Form and establish appropriate resize behaviour using their Attach property.

Example

```
'F'⎕WC'Form' 'TabControl: Default'('Size' 25 50)
'F.TC'⎕WC'TabControl'

'F.TC.IL'⎕WC'ImageList'
'F.TC.IL.'⎕WC'Icon'('' 'APLIcon')
'F.TC.IL.'⎕WC'Icon'('' 'FUNIcon')
'F.TC.IL.'⎕WC'Icon'('' 'EDITIcon')

'F.TC'⎕WS'ImageListObj' 'F.TC.IL'

'F.TC.T1'⎕WC'TabButton' 'One'('ImageIndex' 1)
'F.TC.T2'⎕WC'TabButton' 'Two'('ImageIndex' 2)
'F.TC.T3'⎕WC'TabButton' 'Three'('ImageIndex' 3)

'F.TC.S1'⎕WC'SubForm'('TabObj' 'F.TC.T1')
'F.TC.S2'⎕WC'SubForm'('TabObj' 'F.TC.T2')
'F.TC.S3'⎕WC'SubForm'('TabObj' 'F.TC.T3')
```
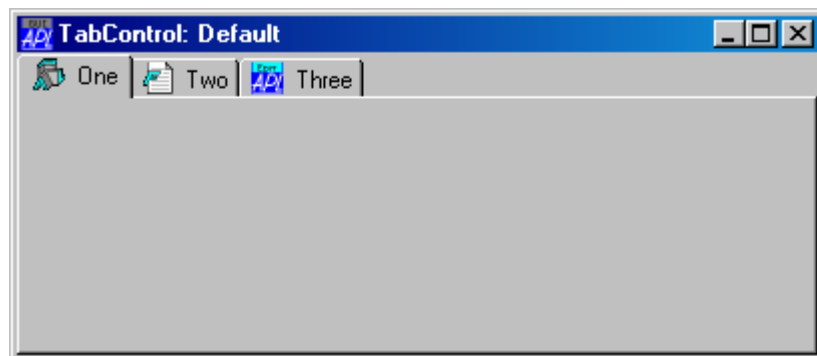
A TabControl object with Style `'Buttons'` or `'FlatButtons'` may be used in a similar way (i.e. to display a set of alternative pages), although buttons in this type of TabControl are more normally used to execute commands. For this reason, these styles of TabControl are borderless.

If Style is `'FlatButtons'`, the FlatSeparators property specifies whether or not separators are drawn between the buttons. The default value of FlatSeparators is 0 (no separators).

The Align property specifies along which of the 4 edges of the TabControl the tabs or buttons are arranged. Align also controls the relative positioning of the picture and Caption within each TabButton. Align may be Top (the default) , Bottom, Left or Right.

If Align is `'Top'` or `'Bottom'`, the tabs or buttons are arranged along the top or bottom edge of the TabControl and picture is drawn to the left of the Caption.

If Align is `'Left'`, the tabs or buttons are arranged top-to-bottom along the left edge of the TabControl, and the pictures are drawn below the Captions.

If Align is `'Right'`, the tabs are arranged top-to-bottom along the right edge of the TabControl, and the pictures are drawn above the Captions.

The MultiLine property determines whether or not your tabs or buttons will be arranged in multiple flights or multiple rows/columns.

The default value of MultiLine is 0, in which case, if you have more tabs or buttons than will fit in the space provided, the TabControl displays an UpDown control to permit the user to scroll them. If MultiLine is set to 1, the tabs are displayed in multiple flights or the buttons are displayed in multiple rows.

The ScrollOpposite property specifies that unneeded tabs scroll to the opposite side of a TabControl, when a tab is selected. Setting ScrollOpposite to 1 forces MultiLine to 1 also.

If MultiLine is 1, the way that multiple flights of tabs or rows/columns of buttons are displayed is further defined by the Justify property which may be `'Right'` (the default) or None.

If Justify is `'Right'` (which is the default), the TabControl increases the width of each tab, if necessary, so that each row of tabs fills the entire width of the tab control. Otherwise, if Justify is empty or `'None'`, the rows are ragged.

By default, the size of the tabs may vary from one to another. Fixed size tabs may be obtained by setting the TabSize property.

To obtain fixed sized tabs with MultiLine set to 1, you must however also set Justify to *'None'*.

If fixed size tabs are in effect, the positions at which the picture and Caption are drawn within each TabButton is controlled by the TabJustify property which may be *'Centre'*, *'Edge'*, or *'IconEdge'*.

The font used to draw the captions in the TabButton objects is determined by the FontObj property of the TabControl.

You cannot specify the foreground or background colours of the tabs/buttons, nor can you use different fonts in different tabs/buttons. The orientation of the Caption text is always determined by the value of the Align property of the TabControl.

The TabObj property is read-only and reports the name of the TabButton that is currently selected.

The MultiSelect property specifies whether or not the user can select more than one button in a TabControl at the same time, by holding down the Ctrl key when clicking. The default is 0 (only one button may be selected). MultiSelect is ignored if Style is *'Tabs'*.

The TabFocus property specifies the focus behaviour for the TabControl object and may be *'Normal'*(the default), *'Never'* or *'ButtonDown'*.

The HotTrack property specifies whether or not the tabs or buttons are automatically highlighted by the mouse pointer. The default is 0 (no highlighting).

The Attach property determines how the TabControl responds to its parent being resized and the default value is *'None'   'None'   'None'   'None'*. This causes the TabControl to maintain its original proportions when its parent is resized.

TabFocus   Property

Applies to   TabControl

The TabFocus property specifies the focus behaviour for the TabControl object.

TabFocus is a character vector that may be *'Normal'* (the default), *'Never'* or
'ButtonDown'.

If TabFocus is *'Normal'*, the tabs or buttons in a TabControl do not immediately
receive the input focus when clicked, but only when clicked a second time. This means
that, normally, when the user circulates through the tabs, the input focus will be given to
the appropriate control in the associated SubForm. However, if the user clicks twice in
succession on the same tab or button, the TabControl itself will receive the input focus.

If TabFocus is *'ButtonDown'*, the tabs or buttons in a TabControl receive the input
focus when clicked.

If TabFocus is *'Never'*, the tabs or buttons in a TabControl *never* receive the input
focus. This allows the user to circulate through a set of tabbed SubForms without ever
losing the input focus to the TabControl itself.

TabJustify   Property
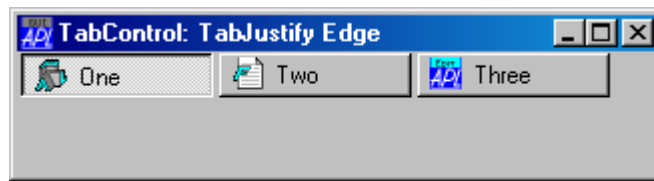
Applies to   TabControl

The TabJustify property specifies, the positions at which the picture and caption are
drawn within each tab or button implemented by a TabButton in a TabControl object.

TabJustify is a character vector that may be *'Centre'*, *'Edge'*, or *'IconEdge'*.
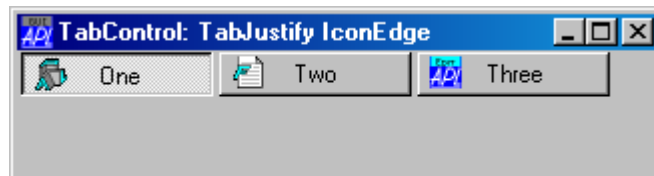Its default value is *'Centre'*.

If TabJustify is *'Centre'*, the picture and caption are arranged in the centre of the
TabButton.
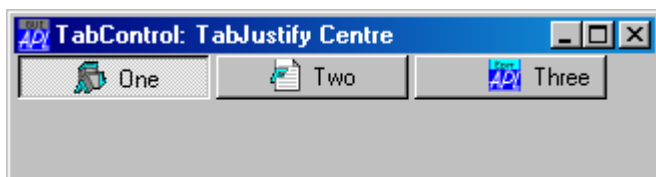
If TabJustify is '*Edge*', the picture and caption are together aligned to the appropriate edge of the TabButton according to the value of Align.

If TabJustify is set to '*IconEdge*', the caption is drawn centrally and only picture is aligned to the edge.

TabJustify is only honoured if fixed size tabs are in effect.

TabObj   Property

Applies to   SubForm, TabBar, TabBtn, TabButton, TabControl

TabObj is a character vector. For a SubForm, it specifies the name of a TabBtn or TabButton object that is associated with the SubForm. Selecting the TabBtn or TabButton causes the SubForm to be given the input focus.

For TabBtn and TabButton objects, TabObj is a read-only property that contains the name of the associated SubForm.

For a TabBar or TabControl, TabObj is a read-only property that contains the name of the currently selected TabBtn or TabButton.

TabSize   Property

Applies to   TabControl

The TabSize property specifies the size of fixed size tabs or buttons in a TabControl object.

By default, the size of the tabs may vary from one to another. Fixed size tabs may be obtained by setting the TabSize property.

TabSize is a 2-element numeric vector that specifies the height and width of the tab. Either or both elements of TabSize may be set to 0 which means "default".

To obtain fixed sized tabs with MultiLine set to 1, you must however also set the Justify property to *'None'*.

If MultiLine is 1 and Justify is *'Right'*, TabSize is ignored.

Target   Property

Applies to   BrowseBox

The Target property is a read-only character string that specifies the chosen folder or other resource selected by the user in a BrowseBox object.

If the BrowseFor is *'Directory'*, Target will contain a directory name followed by the character "\". Otherwise, Target just contains the name.

TargetState   Property

Applies to   TCPSocket

The TargetState property reflects the intended final state of a TCPSocket object. Its possible values are as follows:

Stream	UDP
Client	Open
Server	Bound
Closed	Closed

Setting TargetState to Closed is the recommended way to close a socket. It informs APL that you want the socket to be closed, but only when it is safe to do so. When all the data has been sent, the TCPSocket will generate a TCPClose event which, unless a callback function decides otherwise, will cause the TCPSocket object to disappear.

To control socket closure, you may execute the following steps:

       1. Set TargetState to Closed

       2. **either:**

              a) continue processing

  or: b) wait (using *⎕DQ*) for the TCPSocket to disappear

  or: c) wait (using *⎕DQ*) for the TCPClose event

TCPAccept   Event 371

Applies to   TCPSocket

If enabled, this event is reported when a client connects to a server TCPSocket object.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	`'TCPAccept'` or 371
[3] Socket handle:	an integer

The socket handle reported by this event is the socket handle for the original listening socket that was associated with the TCPSocket before the client connected.

If you want your server to remain available for other clients, you must create a new TCPSocket object in a callback function attached to this event. The new TCPSocket object must be created by cloning the original listening socket. This is done by specifying the socket handle as the value of its SocketNumber property. You may not specify any other properties (except Event and Data) in the $\square WC$ statement that creates the new clone object.

The default processing for this event is to close the socket handle reported by the 3rd element of the event message **unless** it has been associated with a new TCPSocket object by the callback function as described above. You may prevent this from occurring by returning 0 from a callback function. This may be necessary in a multithreaded application.

You may not call TCPAccept as a method or generate this event artificially using $\square NQ$.

TCPClose   Event 374

Applies to   TCPSocket

If enabled, this event is reported when the remote end of a TCP/IP connection breaks the connection.

You may not disable or nullify the operation by setting the action code for the event to ¯1 or by returning 0 from a callback function. You may also not call TCPClose as a method or generate this event artificially using *⎕NQ*.

The event message reported as the result of *⎕DQ*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

     [1] Object: ref or character vector

     [2] Event name or code: *'TCPClose'* or 374

TCPConnect   Event 372

Applies to   TCPSocket

If enabled, this event is reported when a server accepts the connection of a client TCPSocket object and is reported by the client.

You may not disable or nullify the operation by setting the action code for the event to ¯1 or by returning 0 from a callback function. You may also not call TCPConnect as a method or generate this event artificially using *⎕NQ*.

The event message reported as the result of *⎕DQ*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

     [1] Object: ref or character vector

     [2] Event name or code: *'TCPConnect'* or 372

TCPError   Event 370

Applies to   TCPSocket

This event is generated when a fatal TCP/IP error occurs and is reported by a
TCPSocket object.

The default processing for this event is to display a message box containing details of
the TCP/IP error. You may disable the display of this message box by setting the action
code for the event to ¯1 or by returning 0 from a callback function attached to it.

The event message reported as the result of □*DQ*, or supplied as the right argument to
your callback function, is a 3-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	*'TCPError'* or 370
[3] Error code:	a number
[4] Error text:	a character vector

TCPGetHostID   Method 376

Applies to   Root, TCPSocket

This method is used to obtain the IP Address of your PC.

The TCPGetHostID method is niladic.

The result is a character string containing your IP address. If you have more than one, it
will return the first.

Example

```
    TCPGetHostID
193.32.236.43
```

TCPGotAddr   Event 377

Applies to   TCPSocket

If enabled, this event is reported when a host name (specified by the RemoteAddrName or LocalAddrName property) is resolved to an IP address.

You may not disable or nullify the operation by setting the action code for the event to ¯1 or by returning 0 from a callback function. You may also not call TCPGotAddr as a method or generate this event artificially using $\square NQ$.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 2-element vector as follows :

> [1] Object: ref or character vector
>
> [2] Event name or code: *'TCPGotAddr'* or 377

Note that the IP address is not reported in the event message but may be obtained from RemoteAddr or LocalAddr as appropriate.

TCPGotPort   Event 378

Applies to   TCPSocket

If enabled, this event is reported when a port name (specified by the RemotePortName or LocalPortName property) is resolved to a port number.

You may not disable or nullify the operation by setting the action code for the event to ¯1 or by returning 0 from a callback function. You may also not call TCPGotPort as a method or generate this event artificially using $\square NQ$.

The event message reported as the result of $\square DQ$, or supplied as the right argument to your callback function, is a 2-element vector as follows :

> [1] Object: ref or character vector
>
> [2] Event name or code: *'TCPGotPort'* or 378

Note that the port number is not reported in the event message but may be obtained from RemotePort or LocalPort as appropriate.

TCPReady   Event 379

Applies to   TCPSocket

If enabled, this event is reported when the TCP/IP buffers are free and there is no data waiting to be sent in the internal APL queue.

This event is provided to enable you to control the transmission of a large amount of data that cannot be handled in a single call to TCPSend.

The amount of data that the system can handle in one go is limited by TCP/IP buffers, the speed of the network, and the amount of Windows memory and disk space available for buffering.

You may not disable or nullify the operation by setting the action code for the event to ¯1 or by returning 0 from a callback function. However, you *may* call TCPReady as a method or generate this event artificially using *⎕NQ*.

The event message reported as the result of *⎕DQ*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

> [1] Object: ref or character vector
>
> [2] Event name or code: *'TCPReady'* or 379

TCPRecv   Event 373

Applies to   TCPSocket

If enabled, this event is reported when data is received by a TCPSocket object.

You may not disable or nullify the operation by setting the action code for the event to ¯1 or by returning 0 from a callback function. You may also not call TCPRecv as a method or generate this event artificially using *⎕NQ*.

The event message reported as the result of *⎕DQ*, or supplied as the right argument to your callback function, is a 5-element vector as follows :

[1] Object:		ref or character vector
[2] Event name or code:		*'TCPRecv'* or 373
[3] Data:		the data received
[4] IP address:		character vector
[5] Port number:		integer

Elements [4-5] refer to the IP address and port number of the remote process that sent the data.

If the SocketType is *'Stream'*, elements [4-5] will be identical to the values of the RemoteAddr and RemotePort respectively.

If the SocketType is *'UDP'* and there is potentially more than one partner sending you data, the IP address and port number information provided by the TCPRecv event is more reliable than the current values of RemoteAddr and RemotePort as these may already have changed.

TCPSend   Method 375

Applies to   TCPSocket

This method is used to send data to a remote process connected to a TCPSocket object.

The argument to TCPSend is a 1 or 3-element array as follows:

> [1] Data: the data to be sent
>
> [2] IP address: character vector
>
> [3] Port number: integer

If Style is *'Char'*, the data to be sent must be a character vector. If Style is *'Raw'*, the data to be sent must be an integer vector whose elements are in the range -128 to 255. If Style is *'APL'*, any array may be transmitted.

The optional *IP address* and *Port number* parameters specify the intended recipient of the message and apply only if the SocketType is *'UDP'*, in which case they are mandatory. If the SocketType is *'Stream'*, these parameters will be ignored and should be omitted.

TCPSendPicture   Method 380

Applies to   TCPSocket

This method is used to transmit a picture represented by a Bitmap object to a TCP/IP socket. The picture may be transmitted in uncompressed GIF or in PNG format.

The argument to TCPSendPicture is a 1 or 2-element array as follows:

      [1] Bitmap name: character vector

      [2] Picture format: character vector, *'GIF'* or *'PNG'*

If *Picture format* is omitted, the default is GIF format.

Note that the Style of the TCPSocket object must be set to *'Raw'* before you execute the TCPSendPicture method.

The result of the method is an integer that reports the number of bytes that were transmitted.

Example

```
      S1.TCPSendPicture 'BM' 'PNG'
4930
```

Note: Although PNG is recognised as the latest graphics standard for displaying pictures, not all Web browsers support it.

See also: MakeGIF, MakePNG

TCPSocket   Object

Purpose	Provides an interface to Windows sockets (TCP/IP).
Parents	ActiveXControl, Calendar, CoolBand, DateTimePicker, Form, NetType, OLEClient, OLEServer, PropertyPage, Root, SubForm, TCPSocket
Children	Bitmap, BrowseBox, Clipboard, Cursor, FileBox, Font, Form, Icon, ImageList, Locator, Menu, Metafile, MsgBox, OCXClass, OLEClient, Printer, PropertySheet, TCPSocket, Timer, TipField
Properties	Type, LocalAddr, LocalPort, RemoteAddr, RemotePort, Style, Event, LocalAddrName, LocalPortName, RemoteAddrName, RemotePortName, Data, SocketType, SocketNumber, CurrentState, TargetState, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, TCPAccept, TCPClose, TCPConnect, TCPError, TCPGotAddr, TCPGotPort, TCPReady, TCPRecv
Methods	Detach, TCPGetHostID, TCPSend, TCPSendPicture, Wait

The TCPSocket object provides an event-driven mechanism to communicate with other programs (including Dyalog APL) running on other computers, and with the Internet.

The SocketType property is a character vector that specifies the type of the TCP/IP socket. This is either *'Stream'* (the default), or *'UDP'*. SocketType must be defined when the object is created and may not be set or changed using ⎕WS.

The Style property is a character vector that specifies the type of data transmitted or received by the socket; it may be *'Char'*, *'Raw'*, or *'APL'*. The value *'APL'* is valid only if the SocketType is *'Stream'*.

LocalAddr and LocalPort properties identify your end of the connection; RemoteAddr and RemotePort identify the other end of the connection. The values of the two sets of properties are clearly symmetrical; your LocalAddr is your partner's RemoteAddr, and there are strict rules concerning which of them you and your partner may set. See the individual descriptions of these properties for details.

The SocketNumber property is the handle of the socket attached to the TCPSocket object and is generally a read-only property. The only time that SocketNumber may be specified is when a server replicates (clones) a listening socket to which a client has just connected.

Text   Object

Purpose	Displays text.
Parents	ActiveXControl, Animation, Bitmap, Button, Combo, ComboEx, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, Metafile, Printer, ProgressBar, PropertyPage, PropertySheet, RichEdit, Scroll, Spinner, Static, StatusBar, SubForm, TabBar, TipField, ToolBar, TrackBar, TreeView, UpDown
Children	Timer
Properties	Type, Text, Points, FCol, BCol, VAlign, HAlign, Coord, Active, Visible, Event, Dragable, FontObj, OnTop, CursorObj, AutoConf, Data, Translate, Accelerator, KeepOnClose, DrawMode, MethodList, ChildList, EventList, PropList
Events	Close, Create, DragDrop, FontCancel, FontOK, Help, MouseDblClick, MouseDown, MouseMove, MouseUp, Select
Methods	ChooseFont, Detach

The Text object is used to write arbitrary text. It can be used in a Form, SubForm or Group instead of a Label. The main difference is that a Label is implemented as a true window object (thus consuming Windows resources). A Text object is not a window and consumes fewer Windows resources. However, a Label supports DragDrop events and has various useful properties that are not shared by the Text object.

The contents of the Text object are defined by its Text property. This is a character array containing one of the following :

> a simple scalar
> an enclosed vector or matrix (also a scalar)
> a simple vector
> a simple matrix
> a vector of enclosed vectors or matrices

Points is either a simple 2-column matrix of (y,x) co-ordinates, or a 2-element vector of y-coordinates and x-coordinates respectively.

There are two distinct cases :

1)   Points specifies a single point. In this case, Text may be a single scalar
        character, a simple vector, or a matrix containing a block of text. The result is
        that the character, string, or matrix is written at the specified point.

2)   Points specifies more than one point. There are three possibilities :

        a)   If Text is a scalar, its contents are written at each of the points in
                Points. This means that by enclosing a vector or matrix, you can draw
                a string or block of text at several locations.
        b)   If Text is a vector, each element of Text is written at the
                corresponding position in Points.
        c)   If Text is a matrix, each row of Text is written at the corresponding
                position in Points.

FontObj specifies a **single** font to be used to write the Text. See a description of the
FontObj property for details.

FCol specifies the colour of the Text. For a single text item, FCol may be a single
number which specifies one of the standard Windows colours, or a simple 3-element
numeric vector of RGB colour intensities. If more than one text item is involved, FCol
may be a vector which specifies the colour for each item separately. If so, its length must
be the same as the number of points specified by Points.

BCol specifies the background colour of the text, i.e. the colour for the part of the
character cell that is blank. It is defined in the same way as FCol. HAlign and VAlign
specify the horizontal and vertical alignment of the text respectively. They may each be
numeric scalars or vectors with the same length as the number of points specified in
Points. See HAlign and VAlign for details.

When one or more of FCol, BCol, HAlign and VAlign are vectors, the different
components of Text are drawn using the corresponding colours and alignments.

The value of the Dragable property specifies whether or not the Text object can be
dragged by the user. The value of the AutoConf property determines whether or not the
Text object is repositioned when its parent is resized.

Examples

Write '*A*' at (10,20)

```
'g.t1' □WC 'Text' 'A' (10 20)
```

Write '*ħ*' at (10,20) in red

```
'g.t1' □WC 'Text' 'ħ' (10 20) ('FCol' 255 0 0)
```

Write '*Hello*' at (10,20)

```
'g.t1' □WC 'Text' 'Hello' (10 20)
```

Write   '*THIS IS A*
          BLOCK OF
          TEXT   ' at (20,30)

```
BLK←3 9ρ'THIS IS A BLOCK OF TEXT      '
'g.t1' □WC 'Text' BLK (10 20)
```

Write '*A*' at (10,20) and at (30,40)

```
'g.t1' □WC 'Text' 'A' ((10 30)(20 40))
```

Write a red '+' at (10,20) and a green '+' at (20 40)

```
'g.t1' □WC 'Text' '+' ((10 30)(20 40))
            ('FCol' (255 0 0)(0 255 0))
```

Write '*Hello*' at (10,20) and at (30,40)

```
'g.t1' □WC 'Text' (⊂'Hello') ((10 30)(20 40))
```

Write '*A*' at (10,20) and '*B*' at (30,40)

```
'g.t1' □WC 'Text' 'AB' ((10 30)(20 40))
```

Write '*Hello*' at (10,20) and '*World*' at (30,40)

```
'g.t1' □WC 'Text' ('Hello' 'World')
            ((10 30)(20 40))
```

Text   Property

Applies to   Clipboard, Combo, ComboEx, Edit, MsgBox, RichEdit, Spinner,
                      StatusField, Text

This property is associated with the text contents of the Clipboard, Edit, MsgBox and
StatusField objects, or with the edit field in a Combo, or with the contents of a Text
object.

The value of Text is a character array.

In a Combo, StatusField, or a single-line Edit object, Text may be a simple scalar or a
simple vector.

In a multi-line Edit field or in a MsgBox, the value of Text may also be a simple matrix,
or a vector of vectors. If so, "new-line" characters are appended to each row of the
matrix, or to each vector in a vector of vectors, before being displayed. The user may
insert or add a "new-line" character in a multi-line Edit by pressing Ctrl+Enter (Enter
itself is used to press Buttons).

Note that if word-wrapping is in effect in a multi-line Edit object, the structure of Text
does not correspond to the lines displayed.

In a Text object, the value of the Text property may be a simple scalar, an enclosed
vector or matrix, a simple vector, a simple matrix, or a vector of enclosed vectors or
matrices.

In general, the value of Text returned by ⎕WG has the same structure that was assigned to
it by ⎕WC or by the most recent call to ⎕WS. New-Line characters are removed.

You can copy text into the Windows Clipboard by using ⎕WS to set Text for a Clipboard
object. In this case you may specify a simple character scalar, vector or matrix, or a
vector of character vectors. If you are retrieving data from the clipboard that has been
stored by another application, Text will be either a character vector or a vector of
character vectors.

The Text property of a StatusField is updated automatically if its Style property is set to
monitor the status of a key.

TextSize   Property

Applies to   ActiveXControl, Bitmap, Edit, Form, Grid, Printer, Root, Static, StatusBar, SubForm, TabBar, ToolBar

This property has been replaced by the GetTextSize method, which should be used instead. TextSize is retained only for compatibility with previous versions of Dyalog APL.

TextSize is a "read-only" property that reports the size of the bounding rectangle of a text item in a given font. The result is given in the co-ordinate system of the object in question. This property is useful for positioning Text objects.

When you query TextSize you give the text item in whose size you are interested and, optionally, the name of a Font object. The text item may be a simple scalar, a vector or a matrix. If the Font is omitted, the result is given using the current font for the object in question. When you query TextSize on its own, you must enclose the argument to $\Box WG$. This is because APL would otherwise not be able to distinguish between the text string and font name associated with *'TextSize'* and other properties with the same name as these items.

Examples

```
      '.' ⎕WG ⊂'TextSize' 'Hello World'
2.666666746 9.625

      'FNT1' ⎕WC 'Font' 'Arial' 72
      '.' ⎕WG ⊂'TextSize' 'Hello World' 'FNT1'
12 41.875

      '.' ⎕WS 'Coord' 'Pixel'
      '.' ⎕WG ('TextSize' (3 11ρ'Hello World')) 'Coord'
 39 55  Pixel
```

Thumb   Property

Applies to   Form, ProgressBar, Scroll, Spinner, SubForm, TrackBar, UpDown

This property determines and reports the position of the *thumb* in certain objects.

For a Scroll object, the value of Thumb is a single number whose minimum value is 1 and whose maximum value is defined by the Range property.

For a Form or SubForm object, Thumb is a 2-element vector whose elements refer to the position of the thumb in the object's own built-in vertical and horizontal scrollbars respectively.

For other objects, Thumb is a single numeric value in the range defined by the Limits property.

ThumbDrag   Event 440

Applies to   TrackBar

If enabled, this event is generated when the user drags the thumb in a TrackBar object. The event is reported *after* the value of the Thumb property has been updated and is reported continuously as the thumb is dragged. You may not disable this event or alter its effect with a callback function.

The event message reported as the result of $\Box DQ$, or supplied as the right argument to your callback function, is a 3-element vector as follows :

[1] Object:	ref or character vector
[2] Event name or code:	`'ThumbDrag'` or 440
[3] Thumb value:	Integer. The new value of the Thumb property resulting from the user dragging the thumb.

ThumbRect   Property

Applies to   TrackBar

ThumbRect is a *read-only* property that reports the position and size of the bounding rectangle of the thumb in a TrackBar object. It is a 4-element integer vector containing:

[1]   Vertical position of the top-left corner of the bounding rectangle

[2]   Horizontal position of the top-left corner of the bounding rectangle

[3]   Height of the bounding rectangle

[4]   Width of the bounding rectangle

TickAlign   Property

Applies to   TrackBar

TickAlign determines the position of the tick marks in a TrackBar object. For a horizontal TrackBar, TickAlign may be either `'Bottom'` (the default), `'Top'` or `'Both'`. If TickAlign is `'Bottom'`, the ticks are drawn *below* the slider. If TickAlign is `'Top'`, the ticks are drawn above it. If TickAlign is `'Both'`, the ticks are drawn above and below.

For a vertical TrackBar, TickAlign may be either `'Right'` (the default), `'Left'`, or `'Both'` and similarly specifies to which side of the slider bar the ticks are drawn. Note that TickAlign may only be set when the TrackBar is created with `⎕WC` and may not subsequently be altered using `⎕WS`.

Note that ticks are not drawn if the value of HasTicks is 0

TickSpacing   Property

Applies to   TrackBar

The TickSpacing property specifies the spacing between each tick mark in a TrackBar object. It is an integer between 1 and the maximum value of the TrackBar which is defined by the 2nd element of the Limits property.

For example, if you set (*'Limits'* 10 50) and you specify (*'TickSpacing'* 10) you will obtain 5 ticks corresponding to the values 10, 20, 30, 40 and 50 along the slider bar.

Timer   Object

Purpose	To generate an action at regular intervals.
Parents	ActiveXControl, Animation, Bitmap, BrowseBox, Button, Calendar, Circle, Clipboard, Combo, ComboEx, CoolBand, CoolBar, Cursor, DateTimePicker, Edit, Ellipse, FileBox, Font, Form, Grid, Group, Icon, Image, ImageList, Label, List, ListView, Locator, Marker, MDIClient, Menu, MenuBar, MenuItem, Metafile, MsgBox, NetClient, NetType, OLEClient, OLEServer, Poly, Printer, ProgressBar, PropertyPage, PropertySheet, Rect, RichEdit, Root, Scroll, Separator, SM, Spinner, Splitter, Static, StatusBar, StatusField, SubForm, SysTrayItem, TabBar, TabBtn, TabButton, TabControl, TCPSocket, Text, Timer, TipField, ToolBar, ToolButton, ToolControl, TrackBar, TreeView, UpDown
Children	Timer
Properties	Type, Interval, Active, Event, Data, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, Timer
Methods	Detach, Wait

The Timer object is used to generate an event at regular intervals. It can be used to produce animation and to implement "repeaters" such as spin buttons.

The Interval property specifies how often the Timer generates events and is defined in milliseconds. Its default value is 1000.

The Active property determines whether or not the Timer generates events and can be used to switch the Timer off and on as required.

Note that if you create a Timer object whose Timer event generates an error (for example by attaching it to a non-existent callback) it may be very difficult or even impossible to type into the Session, because the error will be displayed over and over again. Care is therefore recommended.

Timer   Event 140

Applies to   Timer

This event is generated at regular intervals by a Timer object and is typically used to fire a callback function to perform a task repeatedly. Returning a 0 from a callback function attached to a Timer event has no effect. The event message reported as the result of ⎕*DQ*, or supplied as the right argument to your callback function, is a 2 element vector as follows :

[1] Object:		ref or character vector
[2] Event code:		'*Timer*' or 140

Tip   Property

Applies to   Animation, Button, Calendar, ColorButton, Combo, ComboEx, DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView, MDIClient, MenuItem, ProgressBar, PropertyPage, PropertySheet, RichEdit, Scroll, SM, Spinner, Static, StatusBar, SubForm, SysTrayItem, TabBar, TabButton, ToolBar, ToolButton, TrackBar, TreeView, UpDown

The Tip property is a character vector or character matrix that specifies a "help" message which is to be displayed when the user positions the mouse pointer over the object. The Tip is displayed in a pop-up TipField object specified by the TipObj property.

`TipField`   Object

Purpose	To display pop-up help.
Parents	ActiveXControl, CoolBand, Form, Group, OLEServer, PropertyPage, PropertySheet, Root, SubForm, TCPSocket
Children	Circle, Ellipse, Font, Marker, Poly, Rect, Text, Timer
Properties	Type, Event, FontObj, FCol, BCol, Data, Translate, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, FontCancel, FontOK
Methods	ChooseFont, Detach

The TipField is used to display pop-up help when the user moves the mouse pointer over an object.

Most of the GUI objects supported by Dyalog APL/W have Tip and a TipObj properties. TipObj specifies the name of a TipField object, and Tip specifies a "help" message. The TipField automatically pops-up to display the Tip when the user moves the mouse pointer over the object. It disappears when the user moves the mouse pointer away.

The TipField is a simple box with a 1-pixel border in which the text specified by Tip is displayed. FCol, BCol and FontObj can be used to customise the appearance of the text within the box. FCol specifies the colour of the text; BCol specifies the background colour with which the box is filled.

If you wish to display Tips for particular objects in different fonts and colours, you must create a separate TipField for each combination of colour and font you need.

TipObj   Property

Applies to   Animation, Button, Calendar, ColorButton, Combo, ComboEx,
DateTimePicker, Edit, Form, Grid, Group, Label, List, ListView,
MDIClient, MenuItem, ProgressBar, PropertyPage, PropertySheet,
RichEdit, Root, Scroll, SM, Spinner, Static, StatusBar, SubForm,
TabBar, ToolBar, TrackBar, TreeView, UpDown

The TipObj property is a character vector that specifies the name of a TipField object in
which the "help" message defined by the Tip property is to be displayed. This message
is displayed when the user positions the mouse pointer over the object.

Note that if TipObj is empty, its value is **inherited** from its parent. Thus setting TipObj
on a Form defines the default TipField (and thus the default appearance of all Tips) for
all the controls in that Form. Setting TipObj on Root defines the default TipField for the
entire application.

TitleHeight   Property

Applies to   Grid

This property is a single number that specifies the height of the column titles displayed
in a Grid object. It is expressed in the units specified by the Coord property of the Grid.

To disable the display of column titles, set TitleHeight to 0.

TitleWidth   Property

Applies to   Grid

This property is a single number that specifies the width of the row titles displayed in a Grid object. It is expressed in the units specified by the Coord property of the Grid.

To disable the display of row titles, set TitleWidth to 0.

Today   Property

Applies to   Calendar, DateTimePicker, NetClient, NetType

The Today property is an IDN that specifies today's date in a Calendar or DateTimePicker object. Its default value is the current date that is set on your computer.

See also CircleToday and HasToday properties.

ToolBar   Object

Purpose	To manage a group of controls such as Buttons.
Parents	ActiveXControl, CoolBand, Form, SubForm
Children	Bitmap, BrowseBox, Button, Calendar, Circle, Combo, ComboEx, Cursor, DateTimePicker, Edit, Ellipse, FileBox, Font, Group, Icon, Image, ImageList, Label, List, ListView, Locator, Marker, Menu, Metafile, MsgBox, OCXClass, Poly, ProgressBar, Rect, RichEdit, Scroll, SM, Spinner, Static, SubForm, Text, Timer, TrackBar, TreeView, UpDown
Properties	Type, Posn, Size, Coord, Align, Border, Active, Visible, Event, VScroll, HScroll, Sizeable, FontObj, FCol, BCol, Picture, OnTop, IconObj, CursorObj, AutoConf, YRange, XRange, Data, Attach, TextSize, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, Translate, Accelerator, AcceptFiles, KeepOnClose, Redraw, MethodList, ChildList, EventList, PropList
Events	Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, Help, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Select
Methods	Animate, Detach, GetFocus, GetTextSize, ShowSIP

The ToolBar object is used to display and manage a set of controls. It is typically used to present a set of Buttons which the user can press to perform various actions. However, the ToolBar has the ability to manage other controls too.

By default, the ToolBar is a raised bar stretched across the top of its parent form. You can alter its appearance using its EdgeStyle property and you can control its alignment with its Align property. Align can be set to Top (the default), Bottom, Left or Right and causes the ToolBar to be attached to the corresponding edge of the Form. A ToolBar aligned Top or Bottom will automatically stretch or shrink horizontally when its parent Form is resized, but it will remain fixed vertically. A ToolBar aligned Left or Right will stretch vertically but will remain fixed horizontally. By default a ToolBar occupies the entire width or length of the side of the Form to which it is attached and is 30 pixels high or wide. You can change these defaults using the Posn and Size properties.

A ToolBar organises its child controls in the order they are created. The way this is done is governed by the value of the Align property. If Align is Top or Bottom, the ToolBar arranges its controls in rows across the screen. If Align is Left or Right, the ToolBar arranges controls in columns.

The first control added to a ToolBar is automatically positioned 2 pixels down and 2 pixels across from its top left corner. The rule for positioning subsequent controls depends upon the value of the Align property.

If Align is `'Top'` or `'Bottom'`, controls are positioned so as to be horizontally adjacent to one another. Whenever a control is added it is positioned relative to the one that immediately preceded it so that its top left corner meets the top right corner of the previous one. The HScroll property determines what happens when the end of the ToolBar is reached. If HScroll is 0 (the default) a control that would otherwise extend beyond the width of the ToolBar is instead positioned immediately below the first control in the ToolBar, thereby starting a new row. Note however that the ToolBar is not automatically resized vertically to accommodate a second row. If you want a multi-row ToolBar you have to set its height explicitly. If HScroll is ¯1 or ¯2, controls continue to be added along the ToolBar even though they extend beyond its right edge and may be scrolled into view using a mini scrollbar. If HScroll is ¯1, the scrollbar is shown whether or not any controls extend beyond the width of the ToolBar. If HScroll is ¯2, the scrollbar appears only if required and may appear or disappear when the user resizes the parent Form.

If Align is `'Left'` or `'Right'`, controls are positioned so as to be vertically adjacent to one another. Whenever a control is added, its top left corner is positioned against the bottom left corner of the previous control. The VScroll property determines what happens when the bottom of the ToolBar is reached. If VScroll is 0 (the default) a control that would otherwise extend beyond the bottom of the ToolBar is instead positioned immediately to the right of the first one; thereby starting a new column. Note however that the ToolBar is not automatically resized horizontally to accommodate a second column. You must set the width of the ToolBar explicitly. If VScroll is ¯1 or ¯2, controls continue to be added down the ToolBar even though they extend beyond its bottom edge and may be scrolled into view using a mini scrollbar. If VScroll is ¯1, the scrollbar is shown whether or not any controls extend beyond the bottom of the ToolBar. If VScroll is ¯2, the scrollbar appears only if required and may appear or disappear when the user resizes the parent Form.

If you specify a value for its Posn property, a control will be placed at the requested position regardless of the value of Style, HScroll or VScroll. However, the next control added will take its default position from the previous one according to the value of these properties. Thus if you wish to group your controls together with spaces between the groups, you need only specify the position of the first one in each group.

ToolboxBitmap   Property

Applies to   ActiveXControl, OCXClass

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

For an OCXClass object, the ToolboxBitmap is a read-only property that reports a bitmap image associated with an OLE Control. This is intended for use by a GUI design tool. Its value is a 2-element vector. The first element is an integer matrix of pixel colours corresponding to the Bits property of a Bitmap object. The second element is a 3-column integer matrix specifying the colour map and corresponds to the CMap property of a Bitmap object.

Thus you can construct a Bitmap object directly from this property with an expression such as:

            '*BM*'⎕*WC'Bitmap'* '','*GAUGE*' ⎕*WG'ToolboxBitmap'*

where *GAUGE* is the name of an OCXClass.

ToolButton   Object

Purpose	The ToolButton object represents a button in a ToolControl.
Parents	ToolControl
Children	Bitmap, Timer
Properties	Type, Caption, Posn, Size, Style, State, Active, Visible, Event, ImageIndex, Data, Hint, HintObj, Tip, Accelerator, Popup, KeepOnClose, MethodList, ChildList, EventList, PropList
Events	Close, Create, Help, MouseDblClick, MouseDown, MouseMove, MouseUp, Select
Methods	Detach

The ToolButton object represents a selectable button in a ToolControl object.

A ToolButton displays a text string, defined by its Caption property, and an image defined by its ImageIndex property. Apart from these characteristics, the appearance of a ToolButton is controlled by its parent ToolControl object.

ImageIndex is an index into an ImageList which contains a set of icons or bitmaps. The ImageList itself is named by the ImageListObj property of the parent ToolControl.

Typically, you will create up to three ImageLists as children of the ToolControl. These will be used to specify the pictures of the ToolButton objects in their normal, highlighted (sometimes termed *hot*) and inactive states respectively. The set of images in each ImageList is then defined by creating unnamed Bitmap or Icon objects as children. Finally, when you create each ToolButton you specify ImageIndex, selecting the three pictures which represent the three possible states of the button.

If you specify only a single ImageList, the *picture* on the ToolButton will be the same in all three states.

The behaviour and appearance of a ToolButton is further defined by its Style property, which may be `'Push'`, `'Check'`, `'Radio'`, `'Separator'` or `'DropDown'`.

Push buttons are used to generate actions and pop in and out when clicked.

Radio and Check buttons are used to select options and have two states, normal (out) and selected (in). Their State property is 0 when the button is in its normal (unselected state) or 1 when it is selected.

Separator buttons are a special case as they have no Caption or picture, but appear as thin vertical grooves used to separate groups of buttons.

A group of adjacent ToolButtons with Style `'Radio'` defines a set in which only one of the ToolButtons may be selected at any one time. The act of selecting one will automatically deselect any other. Note that a group of Radio buttons must be separated from Check buttons or other groups of Radio buttons by ToolButtons of another Style.

A ToolButton with Style `'DropDown'` has an associated popup Menu object which is named by its Popup property. There are two cases to consider.

1. If the ShowDropDown property of the parent ToolControl is 0, clicking the ToolButton causes the popup menu to appear. In this case, the ToolButton itself does not itself generate a Select event; you must rely on the user selecting a MenuItem to specify a particular action.

2. If the ShowDropDown property of the parent ToolControl is 1, clicking the dropdown button causes the popup menu to appear; clicking the ToolButton itself generates a Select event, but does not display the popup menu.

ToolControl   Object

Purpose	The ToolControl object provides a Windows 95/NT ToolBar.
Parents	ActiveXControl, CoolBand, Form, SubForm
Children	Bitmap, BrowseBox, Button, Combo, ComboEx, Cursor, Edit, FileBox, Font, Group, Icon, Image, ImageList, Label, List, ListView, Locator, Menu, MenuBar, Metafile, MsgBox, OCXClass, ProgressBar, RichEdit, Scroll, SM, Spinner, Static, SubForm, Timer, ToolButton, TrackBar, TreeView, UpDown
Properties	Type, Posn, Size, Style, Align, Visible, Event, ImageListObj, FontObj, Data, Attach, Handle, KeepOnClose, MultiLine, Transparent, Divider, ShowCaptions, ShowDropDown, Dockable, UndocksToRoot, Redraw, MethodList, ChildList, EventList, PropList
Events	Close, Configure, ContextMenu, Create, DockAccept, DockCancel, DockEnd, DockMove, DockRequest, DockStart, DragDrop, DropFiles, DropObjects, Expose, Help, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel
Methods	Animate, Detach, GetFocus, GetTextSize, ShowSIP

The ToolControl object provides an interface to the native Windows ToolBar control and supersedes the Dyalog APL ToolBar object.

The tools on a ToolControl are normally represented by ToolButton objects, but the ToolControl may also act as a parent for other objects, including a MenuBar (see below).

Unlike the ToolBar, the ToolControl fully determines the positioning of its children automatically and this is governed by their order of creation. The Posn property of any child of a ToolControl is therefore read-only. Furthermore, the height of objects in a ToolControl may be no greater than that of a ToolButton in the same ToolControl. This in turn is governed by the sizes of the Font and ImageList in use in that ToolControl.

If a ToolControl is the child of a Form, its position and orientation is defined by its Align property. This property is ignored if the ToolControl is the child of a CoolBand.

The overall appearance of the ToolButton objects displayed by the ToolControl is defined by the Style property of the ToolControl itself, rather than by individual ToolButtons. This may be `'Buttons'`, `'FlatButtons'`, `'List'` or `'FlatList'`. The default is `'FlatButtons'`.

The presence or absence of a recessed line drawn above, below, to the left of, or to the right of the ToolControl is controlled by the Divider property whose default is 1 (show divider).

The MultiLine property specifies whether or not ToolButtons (and other controls) are arranged in several rows (or columns) when there are more than will otherwise fit. If MultiLine is 0 (the default), the ToolControl object clips its children and the user must resize it to bring more objects into view.

The Transparent property specifies whether or not the ToolControl is transparent. If so, the visual effect is as if the ToolButtons (and other controls) were drawn directly on the parent Form.

The ShowCaptions property specifies whether or not the captions of ToolButton objects are drawn. Its default value is 1 (draw captions). ToolButtons drawn without captions occupy much less space and ShowCaptions provides a quick way to turn captions on/off for user customisation.

The ShowDropDown property specifies whether or not a drop-down menu symbol is drawn alongside ToolButtons which have Style `'DropDown'`. ShowDropDown also affects the behaviour of such ToolButton objects when clicked.

As a special case, the ToolControl may contain a MenuBar as its **only** child. In this case, Dyalog APL causes the menu items to be drawn as buttons. Although nothing is done to prevent it, the use of other objects in a ToolControl containing a MenuBar, is not supported.

TrackBar Object

Purpose	The TrackBar object is a slider control that allows the user to enter a value by positioning a pointer (thumb) on a scale.
Parents	ActiveXControl, CoolBand, Form, Grid, Group, PropertyPage, SubForm, ToolBar, ToolControl
Children	Bitmap, Circle, Cursor, Ellipse, Font, Icon, Marker, Poly, Rect, Text, Timer
Properties	Type, Posn, Size, Style, Coord, Border, Active, Visible, Event, Thumb, Step, VScroll, HScroll, Limits, SelRange, Sizeable, Dragable, FontObj, BCol, CursorObj, AutoConf, Data, Attach, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, Translate, Accelerator, TickAlign, TickSpacing, HasTicks, ShowThumb, TrackRect, ThumbRect, AcceptFiles, KeepOnClose, Redraw, MethodList, ChildList, EventList, PropList
Events	Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, GotFocus, Help, KeyPress, LostFocus, MouseDown, MouseMove, MouseUp, Scroll, Select, ThumbDrag
Methods	Animate, Detach, GetFocus, GetTextSize, ShowSIP

The TrackBar object consists of a window which contains a slider bar, a thumb, and a set of tick marks. The slider in a TrackBar moves in increments that you specify when you create it. For example, if you specify that the TrackBar should have a range of five, the slider can only occupy six positions: a position at the left side of the TrackBar and one position for each increment in the range. Typically, each of these positions is identified by a tick mark. TrackBars can have either a vertical or horizontal orientation. They can have tick marks on either side, both sides, or neither. A selection of different TrackBars is illustrated below.
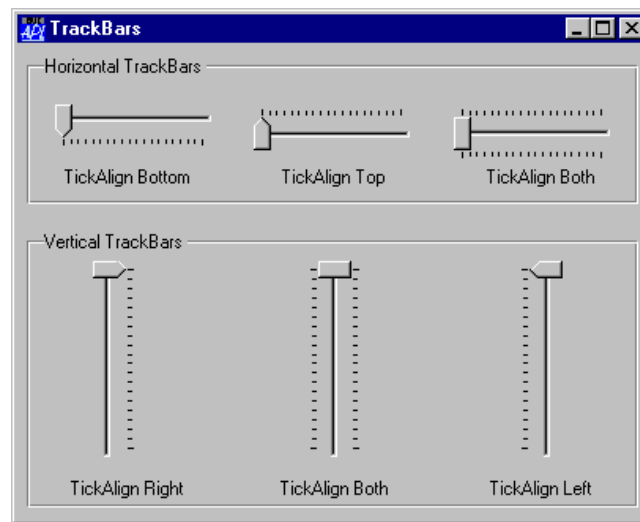
The position and size of the container window are defined by the Posn and Size properties. Its appearance is defined by the EdgeStyle, Border and BCol properties. The defaults are ('*EdgeStyle*' '*None*'), ('*Border*' 0) and ('*BCol*' 0). The default background colour ('*BCol*' 0) obtains either the standard Window Background colour, or grey to match the colour of the parent object if it has a 3-dimensional appearance.

The orientation of a TrackBar is determined by the HScroll and VScroll properties. A horizontal TrackBar is obtained by setting HScroll to ‾1 and VScroll to 0. This is the default. A vertical TrackBar is obtained by setting VScroll to ‾1 and HScroll to 0.

The ShowThumb property determines whether or not the thumb is visible. Its default value is 1. You may toggle this property dynamically using □*WS*.

The TrackBar optionally displays tick marks at the two ends of the slider bar and spaced out along it. This behaviour is determined by the HasTicks property which may be 1 (the default) or 0 and may be set *only* when the object is created by □*WC*.

If HasTicks is 1, the position and frequency of the tick marks is determined by the TickAlign and TickSpacing properties. Note that TickAlign may only be set when the TrackBar is created with $\square WC$ and may not be altered using $\square WS$.

The slider and tick marks in a horizontal TrackBar are drawn along the top of the enclosing window. The slider and tick marks in a vertical TrackBar are drawn along the left edge of the window. The position and size of the slider and the thumb may be obtained from the TrackRect and ThumbRect properties which report these values in pixels. These are *read-only* properties and cannot not be set with $\square WC$ or $\square WS$.

The value of the TrackBar is determined by its Thumb property which is an *integer* that may be set with $\square WS$ or retrieved with $\square WG$. The Limits property specifies the minimum and maximum values of Thumb corresponding to its position at the two ends of the slider bar. The Step property is a 2-element integer vector defining the small and large increments by which the Thumb moves. A small step is obtained by pressing a cursor movement key; a large step is achieved by clicking the left mouse button either side of the thumb or by pressing Page Up and Page Down. The user may also drag the thumb to a new position or move it directly to either end of the slider by pressing Home or End.

An alternative form of the TrackBar is obtained by setting the Style property to `'Selection'`. This may only be done when the object is created using $\square WC$. This style of TrackBar has a slider that is represented by a recessed thick rectangle instead of a solid line. Furthermore, you can select a range of values within the TrackBar by setting the SelRange property. This causes the TrackBar to display a solid bar within the slider and to show the corresponding tick marks as small triangles. Note that there is no way for the user to change SelRange directly; you can only do this using $\square WS$.

In addition to the normal mouse events, the TrackBar generates a Scroll and ThumbDrag event. The Scroll event is the same event that is generated by a Scroll object and is reported when the user repositions the thumb. If enabled, the ThumbDrag event is reported continuously as the user drags the Thumb with the mouse and may be used to synchronise the display of a corresponding value in another object.

TrackRect   Property

Applies to   TrackBar

TrackRect is a *read-only* property that reports the position and size of the bounding rectangle of the slider in a TrackBar object. It is a 4-element integer vector containing:

[1]   Vertical position of the top-left corner of the bounding rectangle

[2]   Horizontal position of the top-left corner of the bounding rectangle

[3]   Height of the bounding rectangle

[4]   Width of the bounding rectangle

Translate   Property

Applies to   ActiveXControl, Animation, Bitmap, BrowseBox, Button, Clipboard, ColorButton, Combo, ComboEx, DateTimePicker, Edit, Form, Grid, Group, ImageList, Label, List, ListView, MDIClient, Menu, MenuBar, MenuItem, Metafile, OCXClass, Printer, ProgressBar, PropertyPage, PropertySheet, RichEdit, Root, Scroll, Separator, Spinner, Static, StatusBar, StatusField, SubForm, SysTrayItem, TabBar, TabBtn, Text, TipField, ToolBar, TrackBar, TreeView, UpDown

This property specifies whether or not character data is to be translated. Translate is a character vector whose values may be `'Inherit'`, `'Translate'`, `'None'`, or `'ANSI'`

A value of `'Translate'` means that all character property values and event parameters are translated to and from $\Box AV$ using the **current** output translation table (normally WIN.DOT).

A value of `'ANSI'` means that all character property values and event parameters are translated to and from $\Box AV$ using the **default** translation scheme as represented by the standard, unmodified, output translation table (WIN.DOT). This is the default value for a run-time application. Unless you require a non-standard translation, it is therefore unnecessary to include input and output tables with a run-time application.

A value of `'None'` means that character data is passed between APL and the object with no translation.

A value of '*Inherit*' means that the object **inherits** its translation from its parent.

Using the development version of Dyalog APL, the default value for the Root and Printer objects is '*Translate*'. Using the run-time version of Dyalog APL, the default value for the Root and Printer objects is '*ANSI*'. For other objects, the default is '*Inherit*'.

Note that changing Translate does not affect existing property values and it can be used temporarily. For example, if you wish to set a particular property of an object without translation, but require other properties to be translated, you can set Translate to '*None*', set the particular property, and then set Translate back to '*Translate*' or '*Inherit*'.
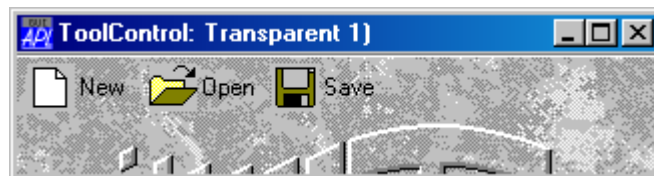
Transparent   Property

Applies to   Animation, ToolControl

The Transparent property specifies whether or not a ToolControl or an Animation has a transparent background.

Transparent is a single number with the value 0 (the default) or 1.

If Transparent is 1, the visual effect is as if the ToolButtons (and other controls owned by the ToolControl) were drawn directly on the parent Form as illustrated below.

TreeView   Object

Purpose	The TreeView object displays a hierarchical list of items.
Parents	ActiveXControl, CoolBand, Form, Group, PropertyPage, SubForm, ToolBar, ToolControl
Children	Bitmap, Circle, Cursor, Ellipse, Font, Icon, ImageList, Marker, Poly, Rect, Text, Timer
Properties	Type, Items, Posn, Size, Coord, Border, Active, Visible, Event, Depth, HasLines, HasButtons, EditLabels, ImageListObj, ImageIndex, SelImageIndex, SelItems, Sizeable, Dragable, FontObj, FCol, BCol, CursorObj, AutoConf, Index, Data, Attach, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, Translate, Accelerator, AcceptFiles, KeepOnClose, CheckBoxes, FullRowSelect, SingleClickExpand, Redraw, MethodList, ChildList, EventList, PropList
Events	BeginEditLabel, Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, EndEditLabel, Expanding, Expose, FontCancel, FontOK, GotFocus, Help, ItemDblClick, ItemDown, ItemUp, KeyPress, LostFocus, MouseDblClick, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, MouseWheel, Retracting, Select
Methods	AddChildren, AddItems, Animate, ChooseFont, DeleteChildren, DeleteItems, Detach, GetFocus, GetItemHandle, GetItemState, GetParentItem, GetTextSize, SetItemImage, SetItemState, ShowItem, ShowSIP

A TreeView object displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files and directories on a disk. Each item consists of a label and an optional bitmapped image, and each item can have a list of sub-items associated with it. By clicking an item, the user can expand and collapse the associated list of sub-items.

The contents of a TreeView object are defined by the Items property; a vector of character vectors that specifies the item labels.

The ImageListObj, ImageIndex and SelImageIndex properties define bitmapped images corresponding to each item. The bitmapped images are drawn to the left of the item labels.

ImageListObj specifies the name of a single ImageList object that contains one or more bitmaps.

ImageIndex and SelImageIndex are $\square IO$ sensitive scalars, or vectors with the same length as the number of items in the object. The value in the i'th element specifies the image for the i'th item and is an index into the corresponding ImageList object. ImageIndex specifies the image displayed when an item is not selected, SelImageIndex specifies the image displayed when an item is selected.

If ImageListObj is specified, but ImageIndex is empty or omitted, the *first* bitmap in the ImageList is drawn alongside every item. If an element of ImageIndex or SelImageIndex specifies a value that does not correspond to a bitmap in the ImageList, no corresponding picture is drawn.

The structure of the items (i.e. the parent/child relationships of the items) is defined by the Depth property. This is either a scalar 0 (the default) which means that all items are *root* items, or it is a numeric vector of the same length as Items. Non-zero values in Depth indicate child items.

The HasLines property is 0, 1 or 2 and determines whether or not lines are drawn that link child items to their corresponding parent item. If HasLines is 0, no lines are drawn. If HasLines is 1, lines are drawn at all except the top level, i.e. the object does not link items at the root of the hierarchy. The default value for HasLines is 2 which provides lines at all levels including the root.

The HasButtons property determines whether or not the TreeView object has a button to the left side of each parent item. It is Boolean with a default value of 1. The user can click the button to expand or collapse the child items as an alternative to double-clicking the parent item. Note that by itself, setting HasButtons to 1 does not add buttons to items at the root of the hierarchy. To achieve this you must also set HasLines to 2.

The CheckBoxes property specifies whether or not check boxes are displayed alongside items in a TreeView.

The FullRowSelect property specifies whether just the item itself, or the entire row of the TreeView, is highlighted when an item is selected. FullRowSelect should not be used if HasLines is 1 or 2

When the user presses the left mouse button over an item, the object generates an ItemDown event. This is followed by an ItemUp event when the mouse button is released. The object also generates an ItemDblClick event when the left mouse is double-clicked over an item. If all three events are enabled, they are reported in the order ItemDown, ItemDblClick, ItemUp.

When a parent item is in its retracted state (its children are not visible) it can be expanded to show its children by the user double-clicking its label or by clicking over its button or tree lines. An Expanding event is reported immediately before the children are shown. Similarly, when a parent item is in its expanded state, it can be retracted to hide its children when a Retracting event is reported.

You can use the Expanding event to define new children for the object just before they are shown. You can also control the actions of these events using callback functions.

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

The SelItems property is a Boolean vector that indicates which item is currently selected. If more items are visible than can fit within the object, a scrollbar is automatically provided. The Index property is a $\Box IO$ sensitive integer that reports the index number of the first item displayed in the object and changes as the items are scrolled.

$Type$ Property

Applies to   ActiveXContainer, ActiveXControl, Animation, Bitmap, BrowseBox, Button, Calendar, Circle, Clipboard, ColorButton, Combo, ComboEx, CoolBand, CoolBar, Cursor, DateTimePicker, Edit, Ellipse, FileBox, Font, Form, Grid, Group, Icon, Image, ImageList, Label, List, ListView, Locator, Marker, MDIClient, Menu, MenuBar, MenuItem, Metafile, MsgBox, OCXClass, OLEClient, OLEServer, Poly, Printer, ProgressBar, PropertyPage, PropertySheet, Rect, RichEdit, Root, Scroll, Separator, SM, Spinner, Splitter, Static, StatusBar, StatusField, SubForm, SysTrayItem, TabBar, TabBtn, TabButton, TabControl, TCPSocket, Text, Timer, TipField, ToolBar, ToolButton, ToolControl, TrackBar, TreeView, UpDown

This property determines the type of an object. Its value is a character vector containing a valid object type. The Type property is set by $\Box WC$ and reported by $\Box WG$, but may not be altered using $\Box WS$.

TypeLibID   Property

Applies to   ActiveXControl, OLEServer

The TypeLibID property is a read-only property that reports the value of the globally unique identifier (GUID) of the Type Library associated with a COM object.

TypeLibFile   Property

Applies to   ActiveXControl, OLEServer

The TypeLibFile property is a read-only property that reports the name of the file in which the Type Library for an COM object is stored.

TypeList   Property

Applies to   OCXClass, OLEClient

This property reports the names of all the special data types defined for a COM object. It is a vector of character vectors returned by `⎕WG`. It may not be set using `⎕WC` or `⎕WS`. Further information about each data type may be obtained using GetTypeInfo.

Note that TypeList reports **all** of the data type names recorded in the Type Library associated with the COM object. If several COM objects are provided within a single .OCX file, the entire set of data types reported may not necessarily be applicable to the Control in question.

Underline   Property

Applies to   Font

This property specifies whether or not the characters in the font associated with a Font object are underlined or not. It is either 0 (normal) or 1 (underlined). There is no default; the value of this property reflects the underline characteristic of the font allocated by Windows.

Undo   Method 170

Applies to   Grid

This method is used to undo the previous change in a Grid object.

The Grid object maintains a buffer of the most recent 8 changes made by the user since the Values property was last set by `⎕WC` or `⎕WS`.

Your application can restore these changes using the Undo method. This restores the specified number of changes made by the user and removes them from the undo stack.

It is therefore not possible to "redo an undo".

The argument to Undo is ⍬,or a single item as follows :

>   [1] Number of changes   integer

If called with an argument of ⍬, the default value for the *Number of changes* is 1. This restores the most recent change.

UndocksToRoot   Property

Applies to   CoolBand, Form, SubForm, ToolControl

Specifies the parent adopted by an object when its Type changes to a Form as a result of an undocking operation.

UndocksToRoot is a single number with the value 0 or 1.

If UndocksToRoot is 1, the object becomes a Form that is a child of Root and therefore becomes completely independent of the Form in which it was previously docked.

If UndocksToRoot is 0, the object becomes a Form that is a child of the Form in which it was previously docked and is therefore always displayed on top of it. This setting is appropriate for a dockable ToolControl.

The default value of UndocksToRoot is 1 if the object was originally created as a child of Root; otherwise it is 0.

UpDown Object

Purpose	The UpDown object is a pair of arrow buttons used to increment or decrement a value.
Parents	ActiveXControl, CoolBand, Form, Group, PropertyPage, SubForm, ToolBar, ToolControl
Children	Bitmap, Circle, Cursor, Ellipse, Font, Icon, Marker, Poly, Rect, Text, Timer
Properties	Type, Posn, Size, Coord, Border, Active, Visible, Event, Thumb, Step, VScroll, HScroll, Wrap, Limits, Sizeable, Dragable, FCol, BCol, CursorObj, AutoConf, Data, Attach, EdgeStyle, Handle, Hint, HintObj, Tip, TipObj, Translate, Accelerator, AcceptFiles, KeepOnClose, Redraw, MethodList, ChildList, EventList, PropList
Events	Close, Configure, ContextMenu, Create, DragDrop, DropFiles, DropObjects, Expose, Help, MouseDown, MouseEnter, MouseLeave, MouseMove, MouseUp, Select, Spin
Methods	Animate, Detach, GetFocus, GetTextSize, ShowSIP

An UpDown object is a pair of arrow buttons that the user can click to increment or decrement a value, such as a scroll position or a number displayed in a companion control. The Spinner object is actually a composite object consisting of an UpDown and a companion Edit.

UpperCase   Property

Applies to   Root

This property specifies whether or not property names returned by `⎕WG` and event names supplied by `⎕DQ` and `⎕NQ` are converted to uppercase or not. It is a Boolean property where 1 means convert to upper case and 0 means not. The default is 0. For example :

```
      '.' ⎕WG 'Type'
Root

      '.' ⎕WS 'UpperCase' 1
      '.' ⎕WG 'Type'
ROOT
```

In Dyalog APL Version 6, property names were always reported in upper case. This was changed in Version 7. The UpperCase property is provided to enable applications developed prior to Dyalog APL/W Version 7 to function with minimal alteration.

ValidIfEmpty   Property

Applies to   Edit, Spinner

This property applies to an Edit object with Style Single and specifies whether or not an empty field is considered to be valid. It also applies to a Spinner. Its value is either 0 (an empty field is not valid) or 1 (an empty field is valid. If the FieldType is Numeric, LongNumeric, Currency, Date or Time, the default value for ValidIfEmpty is 0. Otherwise, its default value is 1.

If ValidIfEmpty is 0 and the user attempts to *leave* the Edit object by shifting the input focus to another control, or by selecting a Button or MenuItem, the Edit object will generate a BadValue event. The Text property will reflect the appearance of the field and be empty, but the Value property will not be changed.

If ValidIfEmpty is 1 and the FieldType is Numeric, LongNumeric, Currency, date or Time, the Value property will be set to θ when the user clears the field and leaves it.

VAlign   Property

Applies to   Text

This property determines the vertical alignment of text in the Text object. It is either a single integer value, or, if the Text object has several components, a corresponding vector of such values. These may be :

0   :   base aligned (the base line of the character is aligned on the y-co-ordinate specified by the Points property).

1   :   half aligned (the centre of the character is aligned on the y-coordinate specified by the Points property).

2   :   cap aligned (the top of the character is aligned on the y-coordinate specified by the Points property).

3   :   bottom aligned (the bottom of the character cell is aligned on the y-co-ordinate specified by the Points property).

4   :   top aligned (the top of the character cell is aligned on the y-coordinate specified by the Points property). This is the default.

Value   Property

Applies to   Edit, Label, Spinner

This property specifies or reports the numeric value associated with an Edit, Label or Spinner object whose FieldType property is set to Numeric, LongNumeric, Date, LongDate or Time.

If the FieldType is Numeric or LongNumeric, the Value property contains a scalar number. If the FieldType is Date or LongDate, the Value property is an integer representing the date as the number of days since 1st January 1900. If the FieldType is Time, the Value property is an integer that contains the number of seconds since midnight.

Values   Property

Applies to   Grid

This property specifies the data values for the cells in a Grid object. Values must be a matrix whose elements are either single numbers, character scalars, character vectors or character matrices. This property is updated as the user moves around the Grid changing data.

VariableHeight   Property

Applies to   CoolBar

The VariableHeight property specifies whether or not a CoolBar displays bands at the minimum required height, or all the same height (that of the largest).

VariableHeight is a single number with the value 0 (same height) or 1 (variable height). The default is 1.

View   Property

Applies to   ListView

The View property specifies how the items in a ListView object are displayed. It is a character vector which may have one of the following values; `'Icon'` (the default), `'SmallIcon'`, `'List'` or `'Report'`.

When View is `'Icon'` or `'SmallIcon'`, the items are arranged *row-wise* with large or small icons as appropriate. When View is set to `'List'`, the items are arranged *column-wise* using small icons. When View is set to `'Report'`, the items are displayed in a single column using small icons but with the matrix specified by ReportInfo displayed alongside. In this format, the ListView also provides column headings which are specified by the ColTitles property. The alignment of these titles (and of the data in the columns beneath them) is defined by the ColTitleAlign property. Examples of different views are illustrated below.

Visible   Property

Applies to   ActiveXControl, Animation, Button, Calendar, Circle, ColorButton,
Combo, ComboEx, CoolBand, DateTimePicker, Edit, Ellipse, Form,
Grid, Group, Image, Label, List, ListView, Marker, MenuBar, Poly,
ProgressBar, PropertySheet, Rect, RichEdit, Scroll, SM, Spinner,
Splitter, Static, StatusBar, StatusField, SubForm, TabBar, TabBtn,
Text, ToolBar, ToolButton, ToolControl, TrackBar, TreeView,
UpDown

This property specifies whether or not an object is currently visible. It is a single number
with the value 0 (object is invisible) or 1 (object is visible). The default is 1. Setting
Visible on and off is a way to pop a dialog box up and down as required.

Note that an invisible object is not necessarily inactive, and is capable of generating
events. For example, a Button with a Cancel property of 1 will generate a Select (30)
event (if enabled) whether or not it is visible. An invisible object will also respond to
methods and events sent to it by *⎕NQ*.

VScroll   Property

Applies to   Combo, ComboEx, Edit, Form, Grid, List, ListView, RichEdit, Scroll, StatusBar, SubForm, TabBar, ToolBar, TrackBar, UpDown

This property typically determines whether or not an object has a vertical scrollbar. It is a single integer with the value ¯3, ¯2, ¯1, or 0.

For a Form object, the value ¯1 specifies that the Form has a vertical scrollbar. A value of 0 (which is the default) means that it does not.

When applied to an Edit object, the value ¯2 specifies that the data is scrollable vertically, but only by using the cursor keys; a scrollbar is not provided. A value of ¯1 causes a scrollbar to be displayed (whether or not one is needed).

When applied to a List object, a value of ¯1 or ¯2 causes a scrollbar to be displayed if required (when the list of items exceeds the height of the object).

When applied to a Combo or ComboEx object, a value of ¯1 or ¯2 causes a scrollbar to be displayed, whether or not one is required..

For all three object, a value of 0 inhibits scrolling altogether.

For a Scroll object, VScroll may be ¯1 or 0. If it is ¯1, the direction of the scrollbar is vertical. If both HScroll and VScroll are set to ¯1, HScroll takes precedence and forces VScroll back to 0.

For a StatusBar, TabBar or ToolBar with Align set to Left or Right, VScroll determines whether or not a vertical scrollbar is provided and how the object positions its children. If VScroll is 0 (the default) the object organises its children in multiple columns and does not provide a scrollbar. If VScroll is ¯1 or ¯2, the object organises its children in a single column and provides a mini scrollbar to allow those positioned beyond the bottom edge of the object to be scrolled into view. If VScroll is ¯1, the scrollbar is always shown. If VScroll is ¯2, it is only shown when needed.

For a Grid, VScroll may be 0 (no vertical scrollbar), ¯1 (scrollbar is displayed when required), ¯2 (same as ¯1) or ¯3 (scrollbar is always displayed).

VScroll   Event 38

Applies to   Form, SubForm

If enabled, this event is generated when the user attempts to move the thumb in a vertical scrollbar in a Form or SubForm. This event occurs only in a Form whose VScroll property is set to ¯1 and is distinct from the Scroll event which is generated by a Scroll object. The event may be generated in one of three ways :

a)   dragging the thumb

b)   clicking in one of the "arrow" buttons situated at the ends of the scrollbar. This is termed a small change, the size of which is defined by Step[1].

c)   clicking in the body of the scrollbar. This is termed a large change, the size of which is defined by Step[2].

The event message reported as the result of □DQ, or supplied as the right argument to your callback function, is a 4-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	`'VScroll'` or 38
[3] Scroll Type:	numeric
[4] Position:	numeric

The value of Scroll Type is 0 (drag), 1 or ¯1 (small change) or 2 or ¯2 (large change). The sign indicates the direction. The value of Position is the new (requested) position of the thumb. Notice however that the event is generated **before** the thumb is actually moved. If your callback function returns a scalar 0, the position of the thumb will remain unaltered.

Wait — Method 147

Applies to   BrowseBox, Clipboard, FileBox, Form, Locator, Menu, MsgBox, PropertySheet, Root, SysTrayItem, TCPSocket, Timer

The Wait method is the same as executing `⎕DQ` on the object.

The Wait method is niladic.

```
        'F'⎕WC'Form'
...
        Z←F.Wait
```

WantsReturn — Property

Applies to   Edit, RichEdit

This Boolean property specifies the behaviour of the Enter key for a multi-line Edit (Style `'Multi'`) and a RichEdit object.

A value of 0 means that the Enter key is ignored by the Edit or RichEdit. Instead, the Enter key will (if appropriate) cause a Select event on a Button in the same Form. The user must press Ctrl+Enter to input a new line. A value of 1 means that pressing the Enter key will introduce a new line into the object.

WantsReturn must be established when the object is created by `⎕WC` and may not subsequently be altered using `⎕WS`. Its default value is 0 in an Edit and 1 in a RichEdit.

WeekNumbers — Property

Applies to   Calendar, DateTimePicker

The WeekNumbers property specifies whether or not a Calendar object displays week numbers.

WeekNumbers is a single number with the value 0 (week numbers are *not* shown) or 1 (week numbers *are* shown); the default is 0.

Weight   Property

Applies to   Font

This property specifies the degree of boldness of a font associated with a Font object. It is a number in the range 0 to 1000, where 0 represents *very feint* and 1000 represents *very bold*. There is no default; the value of this property reflects the degree of boldness of the font allocated by Windows. In general, 400 means *normal* and 700 means *bold*.

WinIniChange   Event 133

Applies to   Root

If enabled, this event is reported when another application updates WIN.INI (Version 7) or changes relevant registry settings (Version 8) using the standard API calls. The event is reported after the change has taken place and cannot be disabled or inhibited in any way. If your application depends upon WIN.INI or registry settings, this event gives you the opportunity of refreshing these parameters if they are changed.

The event message reported as the result of *⎕DQ*, or supplied as the right argument to your callback function, is a 2-element vector as follows :

[1] Object:	ref or character vector
[2] Event code:	*'WinIniChange'* or 133

WordFormat   Property

Applies to   RichEdit

The WordFormat property is identical to the CharFormat property except that it is used to apply formatting to the selected word or words in a RichEdit object. If the selection is empty but the insertion point is inside a word, the formatting is applied to the word. See CharFormat for further details.

Wrap   Property

Applies to   ListView, ProgressBar, Spinner, UpDown

The Wrap property is Boolean and has a default value of 1.

For a ListView it specifies whether or not long labels (specified by the Items property) may be wrapped or not.

For a ProgressBar object it determines whether or not the object starts over again when it reaches its upper limit.

For Spinner and UpDown objects, Wrap determines what happens when the value in the Spinner reaches its upper or lower limit. If Wrap is 1 the Spinner will wrap around to its opposite limit. Otherwise it will stick.

XRange   Property

Applies to   ActiveXControl, Bitmap, Form, Grid, Group, MDIClient, Metafile, Printer, Root, Static, StatusBar, SubForm, TabBar, ToolBar

XRange and YRange together determine a user-defined co-ordinate system. These properties are effective on the object's children which have Coord set to *'User'*.

XRange is a 2-element numeric vector containing the x-coordinate of the top left and bottom right interior corners of the object respectively. See Coord for further details.

Yield   Property

Applies to   Root

This property specifies the frequency with which APL *yields* to Windows and applies mainly to Version 7. Multi-tasking in Windows 3.x is implemented by task switching between applications whenever any application requests a message from the queue. An application that is purely performing computational or file-handling tasks will therefore prevent all other applications from running. Well-behaved Windows applications should *yield* control by requesting a message periodically even though no user interaction is currently taking place. However, this operation takes a perceptible length of time, even if no other applications are running.

By default, Dyalog APL/W yields control to Windows approximately every 1/5th of a second. This is implemented by checking the time at the beginning of each line of executable APL code and yielding if 1/5th of a second or more has elapsed since the last yield. This mechanism also allows APL to detect user interrupts (which are simply Windows messages) during the execution of code.

In most circumstances yielding every 1/5th of a second produces "good" Windows behaviour with little impact on APL throughput. However, in some cases it may be beneficial to use the Yield property to explicitly control the yield frequency. An example is in an application that takes an appreciable time to redraw an existing graphical picture. If APL yields before the entire picture has been redrawn, some objects will be erased before others are redrawn, causing a flickering effect.

The value of Yield is an integer expressed in 1/1000's of a second. Its default value is 200. Yield defines the period of time allowed to elapse between the execution of successive lines of APL code before APL yields to Windows by requesting a message from the Windows queue. If Yield is set to zero APL does not explicitly yield.

Note that the value of this property only controls the yield frequency when APL is executing user-defined code. APL may also yield implicitly during $\square DL$, $\square DQ$, $\square NQ$, $\square WC$, $\square SR$, $\square WS$ and $\square WG$ and in communicating with Auxiliary processors. Note that setting Yield to 0 (or to a very high value) during the execution of code that does not implicitly yield will effectively de-activate all other applications (including Program manager) and disable APL interrupts (Ctrl+Break). It should therefore be used with extreme caution.

YRange   Property

Applies to   ActiveXControl, Bitmap, Form, Grid, Group, MDIClient, Metafile, Printer, Root, Static, StatusBar, SubForm, TabBar, ToolBar

XRange and YRange together determine a user-defined co-ordinate system. These properties are effective on the object's children which have Coord set to *'User'*.

YRange is a 2-element numeric vector containing the y-coordinate of the top left and bottom right interior corners of the object respectively. See Coord for further details.