



▶ Writing ActiveX Controls

▶ The ActiveXControl Object

## ActiveXControl Object

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

**Parents** Form, Root

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

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

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

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

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

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

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

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

### Dyalog Ltd

South Barn  
Minchens Court  
Minchens Lane  
Bramley  
Hampshire  
RG26 5BH  
United Kingdom

**Phone:**

+ 44 (0) 1256 830 030

**Fax:**

+ 44 (0) 1256 830 031

**e-mail:**

sales@dyalog.com



➤ Writing ActiveX Controls

➤ The ActiveXControl Object

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 "Dialog ", 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  $\underline{A}$  or  $\Delta$ . *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 *3D* properties may be used to control the outline appearance of the ActiveXControl object.

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

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

The *Container* property (see below) 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.

### Dyalog Ltd

South Barn  
Minchens Court  
Minchens Lane  
Bramley  
Hampshire  
RG26 5BH  
United Kingdom

**Phone:**

+ 44 (0) 1256 830 030

**Fax:**

+ 44 (0) 1256 830 031

**e-mail:**

sales@dyalog.com



#### ▶ Writing ActiveX Controls

#### ▶ The ActiveXControl Object

When an instance of an ActiveXControl is created, it generates first a PreCreate event (see below), 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.

## Container Property

The Container property is a read-only property whose value is the `IOR` of an ActiveContainer object (see below) 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 `INS` 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.

## PreCreate Event 534

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.

### Dyalog Ltd

South Barn  
Minchens Court  
Minchens Lane  
Bramley  
Hampshire  
RG26 5BH  
United Kingdom

**Phone:**

+ 44 (0) 1256 830 030

**Fax:**

+ 44 (0) 1256 830 031

**e-mail:**

sales@dyalog.com



▶ **Writing ActiveX Controls**

▶ **The ActiveXControl 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 4-element vector as follows :

- [1] Object name character vector
- [2] Event name or code `'PreCreate'` or 534

### ActiveXContainer Object

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

**Parents** ActiveXControl

**Children** None

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 Font, 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 ActiveXControl to react to such changes.

**Dyalog Ltd**

South Barn  
Minchens Court  
Minchens Lane  
Bramley  
Hampshire  
RG26 5BH  
United Kingdom

**Phone:**

+ 44 (0) 1256 830 030

**Fax:**

+ 44 (0) 1256 830 031

**e-mail:**

sales@dyalog.com



### AmbientChanged Event 533

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 Font, FCol and BCol properties of the ActiveXContainer (see above).

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

The event message reported as the result of `DDQ`, or supplied as the right argument to your callback function, is a 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, Property code and Description may be one of the following:

Property code	Description	Meaning
<code>-701</code>	DISPID_AMBIENT_BACKCOLOR	BCol has changed
<code>-703</code>	DISPID_AMBIENT_FORECOLOR	FCol has changed
<code>-705</code>	DISPID_AMBIENT_FONT	Font has changed
<code>-1</code>	DISPID_AMBIENT_UNKNOWN	unknown

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

#### Dyalog Ltd

South Barn  
Minchens Court  
Minchens Lane  
Bramley  
Hampshire  
RG26 5BH  
United Kingdom

**Phone:**

+ 44 (0) 1256 830 030

**Fax:**

+ 44 (0) 1256 830 031

**e-mail:**

sales@dyalog.com