AutoCAD 2006 VBA

A Programmer's Reference

Joe Sutphin

Apress®

AutoCAD 2006 VBA: A Programmer's Reference

ISBN: 1-59059-579-3

Library of Congress Cataloging-in-Publication data is available upon request.

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit http://www.springeronline.com.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Source Code section.

CHAPTER 4

AutoCAD Events

Events occur as a result of actions happening while your program is running, such as opening or saving a drawing. They allow you to write source code that will execute whenever that event occurs. Messages such as “Would you like to save changes?” are the common results of a user action that has triggered an event.

AutoCAD 2006 supports three levels of events: application, document, and object. These event levels correspond to the three major areas of AutoCAD. Event handlers are `Sub` procedures that are executed automatically every time their associated event occurs. Some AutoCAD events allow information to be passed to the event handlers through parameters.

Application-Level Events

Changes to the AutoCAD application environment result in application-level events. These include the opening and saving of drawings, running of AutoCAD commands, changes to system variables, and changes to the AutoCAD application window.

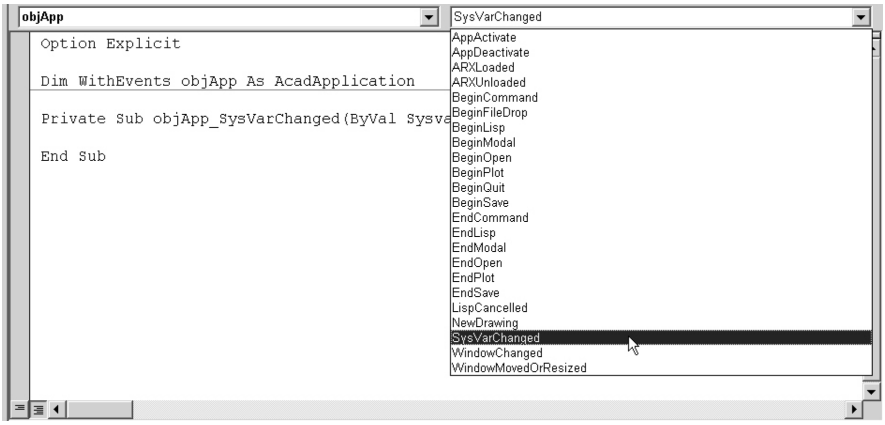
Application-level events aren’t enabled when you load a VBA project. The following example illustrates the steps you need to take to enable application-level events.

First, insert a new class module by selecting Insert ➤ Class Module, and name the new class module appropriately, for example `clsApplicationEvents`. Then declare an object of type `AcadApplication` using the `WithEvents` keyword.

```
Public WithEvents objApp As AcadApplication
```

You should now see the new object `objApp` appear in the Object list box of the class module and all its event procedures are available in the Procedure list box, as shown in Figure 4-1.

You can then go ahead and write the code within these procedures that you want to be executed each time the events occur. However, these event handlers won’t be triggered unless you’ve set the reference to correspond to the `Application` object.



```
Option Explicit

Dim WithEvents objApp As AcadApplication

Private Sub objApp_SysVarChanged(ByVal Sysva

End Sub
```

Figure 4-1. *The Object and Procedure boxes*

You can do this as follows. In the ThisDrawing or any code module, declare a variable to be a new instance of the class module you just created, and in a subroutine set this variable to hold a reference to the Application object.

```
Option Explicit
Public objApp As New clsApplicationEvents

Public Sub InitializeEvents()
  Set objApp.objApp = ThisDrawing.Application
End Sub
```

As soon as the InitializeEvents subroutine is called, in this case by running the App_StartMacro macro shown next, the application-level events are enabled.

```
Public Sub App_StartMacro()
  InitializeEvents
End Sub
```

The following examples illustrate writing code within the event procedures of the class module to execute when those events occur. The first informs the user when a system variable changes, and the second ensures that the AutoSave interval for a new drawing is always set to 30 minutes.

```
Private Sub objApp_SysVarChanged(ByVal SysvarName As String, _
                                 ByVal newVal As Variant)
  MsgBox "The System Variable: " & SysvarName & " has changed to " & newVal
End Sub

Private Sub objApp_NewDrawing()
    ThisDrawing.SetVariable "SAVETIME", 30
    MsgBox "The autosave interval is currently set to 30 mins"
End Sub
```

The following list summarizes the events available at the application level:

`AppActivate`

`AppDeactivate`

`ARXLoaded`

`ARXUnloaded`

`BeginCommand`

`BeginFileDrop`

`BeginLisp`

`BeginModal`

`BeginOpen`

`BeginPlot`

`BeginQuit`

`BeginSave`

`EndCommand`

`EndLisp`

`EndModal`

`EndOpen`

`EndPlot`

`EndSave`

`LispCancelled`

`NewDrawing`

`SysVarChanged`

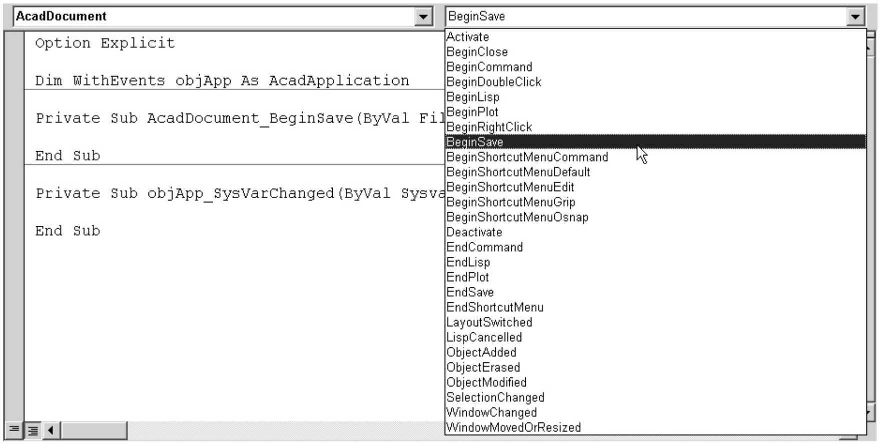
`WindowChanged`

`WindowMovedOrResized`

■ **Note** Unlike Visual LISP, VBA provides no `CommandCancelled` or `CancelledCommand` event to respond to. When a user presses the Esc key, in most cases it won't fire the `EndCommand` event.

Document-Level Events

Changes to a document or its contents result in document-level events. Adding or editing objects and regeneration of the drawing are just some examples of document-level events. Unlike application-level events, document-level events are available by default in the `ThisDrawing` module of an AutoCAD project. If you choose the `AcadDocument` object in the Object list box of the `ThisDrawing` module, the document-level events are listed in the Procedure list box, as shown in Figure 4-2.



```
Option Explicit

Dim WithEvents objApp As AcadApplication

Private Sub AcadDocument_BeginSave(ByVal Fil

End Sub

Private Sub objApp_SysVarChanged(ByVal Sysva

End Sub
```

Figure 4-2. *The Procedure box showing document-level events*

The following is a summary of events available at the document level:

`Activate`

`BeginClose`

`BeginCommand`

`BeginDocClose`

`BeginDoubleClick`

`BeginLisp`

`BeginPlot`

`BeginOpen`

`BeginRightClick`

`BeginSave`

`BeginShortcutMenuCommand`

`BeginShortcutMenuDefault`

```
    BeginShortcutMenuEdit

    BeginShortcutMenuGrip

    BeginShortcutMenuOsnap

    Deactivate

    EndCommand

    EndLisp

    EndOpen

    EndPlot

    EndSave

    EndShortcutMenu

    LayoutSwitched

    LispCancelled

    ObjectAdded

    ObjectErased

    ObjectModified

    SelectionChanged

    WindowChanged

    WindowMovedOrResized
```

Next you'll look at some of the document events and why you might want to add code to execute when they occur.

The BeginCommand and EndCommand Events

When you issue an AutoCAD command such as LINE or DIM, the BeginCommand event is triggered. Any code that you've written inside the BeginCommand event procedure is then executed. Once the code associated with the BeginCommand event has finished executing and after the command itself has finished, the EndCommand event is triggered. Now any code that you've written inside the EndCommand event procedure executes. You may have code associated with either, both, or neither of the events. If a command is canceled prior to completion, such as when a user presses the Esc key, it doesn't fire the EndCommand event.

The following BeginCommand event procedure illustrates creating a layer called Objects (if it doesn't exist) and making the layer active based on the user starting the LINE command:

```
Option Explicit
Public objCurrentLayer As AcadLayer
Public objPreviousLayer As AcadLayer
```

```
Private Sub AcadDocument_BeginCommand(ByVal CommandName As String)
  Set objPreviousLayer = ThisDrawing.ActiveLayer
  Select Case CommandName
    Case "LINE"
      If Not ThisDrawing.ActiveLayer.Name = "OBJECTS" Then
        Set objCurrentLayer = ThisDrawing.Layers.Add("OBJECTS")
        ThisDrawing.ActiveLayer = objCurrentLayer
      End If
  End Select
End Sub
```

The corresponding `EndCommand` event procedure puts the user back to the layer he or she was on if you had to change it in order to draw a line:

```
Private Sub AcadDocument_EndCommand(ByVal CommandName As String)
  Select Case CommandName
    Case "LINE"
        ThisDrawing.ActiveLayer = objPreviousLayer
    End Select
  Set objCurrentLayer = Nothing
  Set objPreviousLayer = Nothing
End Sub
```

The reason for using such code is that it brings continuity to your drawings. For example, all lines will be on the Objects layer, and by adding further code, all text could be added automatically to a Text layer.

The `BeginOpen` and `EndOpen` Events

When AutoCAD receives a request to open an existing drawing file, the `BeginOpen` event is triggered. Once AutoCAD has finished loading the drawing file and it's visible, an `EndOpen` event occurs. One possible use of this event procedure would be to store all the current system variables before you open a drawing. Then once the drawing is opened, you restore the system variables back to their previous values.

When AutoCAD receives a request to create a new drawing file, a slightly different sequence of events occurs. The `BeginOpen` event occurs as before, and then the `BeginSave` event occurs (see the section "The `BeginSave` and `EndSave` Events" for details). This is followed by an `EndSave` event and finally an `EndOpen` event.

The `BeginClose` and `BeginDocClose` Events

The `BeginClose` event is triggered upon the closing of a drawing session within AutoCAD. Be careful when you use this event! If you attempt to perform a lengthy task, it can frustrate users due to slowness, or it could even result in serious problems with AutoCAD, causing it to become unstable, lock up, or crash entirely. The `BeginDocClose` event is similar to the `BeginClose` event. However, the `BeginDocClose` event allows you to cancel the Close command.

The Activate and Deactivate Events

The Activate event is triggered when a drawing window gains focus. When only one drawing is opened in AutoCAD, it will always have focus. When multiple drawings are opened, this event is triggered when switching between drawing windows. The drawing window that loses focus triggers the Deactivate event. Normally, the Deactivate event is triggered just before the Activate event as drawing window focus is switched.

Keep in mind that the Deactivate event indicates the drawing has lost focus. Firing off a procedure as a result of this event might not be a good idea, as it may not complete its task until the drawing regains focus (indicated by another Activate event). You can develop programs to work with what is called a zero document state, meaning there are no drawings opened. Consult the Autodesk developer guide for more information on this topic.

The BeginSave and EndSave Events

Immediately before AutoCAD begins to save the current drawing, the BeginSave event is triggered. Once AutoCAD has completed saving the drawing file, the EndSave event occurs. You might use the BeginSave event to query whether or not the user wants to purge his or her drawing before saving it. You could use the EndSave event to reinitialize standard layers, linetypes, and text styles that may have been purged because they weren't currently being used.

As you can see, AutoCAD has provided some very useful events that greatly enhance its controllability.

Object-Level Events

Object-level events occur when changes are made to a specific entity that you've declared as having events. Modified is the only object-level event and, as you would expect, it occurs when the specified object is modified.

To use object-level events, you must first create a new class module and declare a variable to hold a reference to the object whose Modified event you want to code. You might call the new class module something like clsObjectEvent. The new class module contains the declaration of the object using the VBA keyword WithEvents, as in the following example:

```
Public WithEvents objLine As AcadLine
```

The new object then appears in the Object list box of the class module and the event procedure for the new object may now be written within the class module in the same way as for other subroutines. For your event procedures to be triggered, you must associate the declared object in the class module with the object of interest. For this Line object example, you could do this by placing the following code in the ThisDrawing module or any code module:

```
Dim objLine As New clsObjectEvent
Public Sub InitializeEvent()
Dim dblStart(2) As Double
Dim dblEnd(2) As Double
    dblEnd(0) = 1: dblEnd(1) = 1: dblEnd(2) = 0
    Set objLine.objLine = ThisDrawing.ModelSpace.AddLine(dblStart, dblEnd)
End Sub
```

You first declare your object to be a new instance of the class `clsObjectEvent`, and in the initial event procedure set the `objLine` variable to hold a reference to a newly created `Line` object. Now, as soon as this procedure is called, a new `Line` object is created that responds to any changes made to the line in question by executing the code in the `Modified` event procedure.

So if you put the following code in the `clsObjectEvent` class module, the new coordinates of the `Line` object will be displayed to the user, whenever the line is moved, rotated, scaled, etc.:

```
Private Sub objLine_Modified(ByVal pObject As AutoCAD.IAcadObject)
Dim varStartPoint As Variant
Dim varEndPoint As Variant

    varStartPoint = pObject.StartPoint
    varEndPoint = pObject.EndPoint
    MsgBox "New line runs from (" & varStartPoint(0) & ", " & _
        varStartPoint(1) & ", " & varStartPoint(2) & " ) to (" & _
        varEndPoint(0) & ", " & varEndPoint(1) & ", " & varEndPoint(2) & ")."

End Sub
```

Summary

In this chapter you've seen how AutoCAD events can greatly increase programming flexibility. The steps involved in initializing events may seem a little complex to those first encountering events, but hopefully this won't deter you from taking full advantage of the power and flexibility they provide.