CHAPTER 14

APPLICATION DEPLOYMENT METHODS

This chapter discusses the possible methods in deploying an ArcMap™ based application. When working in the **VBA**, Visual Basic for Applications, environment, the programmer can deploy an application in the form of an ArcMap™ template (*.mxt) or an ArcMap™ document file, which references an ArcMap™ template. In this mode the source for the application is packaged with the ArcMap™ template. To ensure that the source is not modified or viewed, the programmer can password protect the ArcMap™ template. In the **VB**, Visual Basic, environment, the programmer will create DLL(s) for the application.

From a performance point of view, creating DLL(s) for an application will provide slightly better performance over creating an ArcMap™ project file, which references an ArcMap™ template. The downside is that this process is much more involved than creating an ArcMap™ template.

14.1 Creating an ArcMap Template File

The steps presented below describe how the developer can create an ArcMap™ template file. In the **VBA** environment, the developer works within an ArcMap™ document file. This document file contains the VBA code, forms, combo boxes, tools, etc. which comprise the application. Once the application has been tested and is ready for deployment, the developer should:

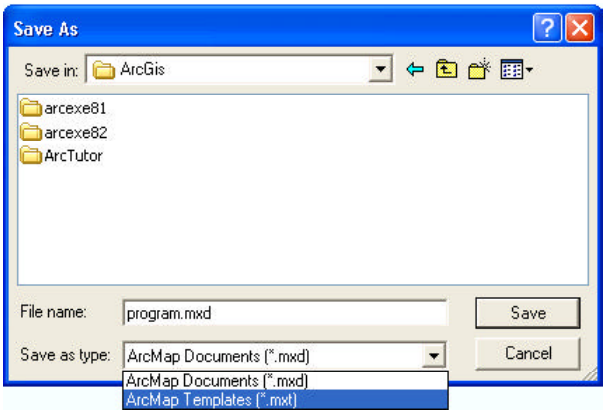


Figure 14-1 Save As Dialog Box

✍ **1** Open the ArcMap™ document file, which is to be converted into an ArcMap™ template file.

Create the ArcMap™ template file

✍ **2** **Click** at the **File** menu and the **Save As...** sub-menu. The conventional file browsing window of Figure 14-1 is displayed.

✍ **3** **Scroll** down in the "Save as type:" combo box, and **click** at the ***.mxt** option.

✍ **4** **Click** in the "File name:" data field, and **enter** the name of the template file to be created. **Click** at the **Save** button to create the template file.

✍ **5** **Click** at the **File** menu and then at the **Exit** sub menu to exit ArcMap™.

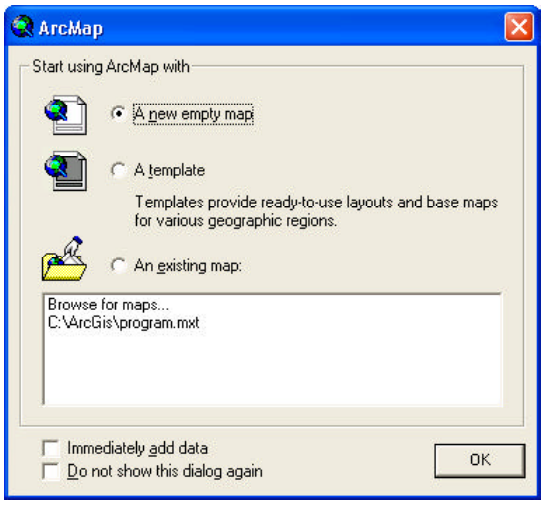


Figure 14-2 ArcMap Initial Dialog Box

Apply password protection to the template file

✍ **6** **Invoke** the ArcMap™ program. The "Start using ArcMap with" selection box of Figure 14-2 is displayed.

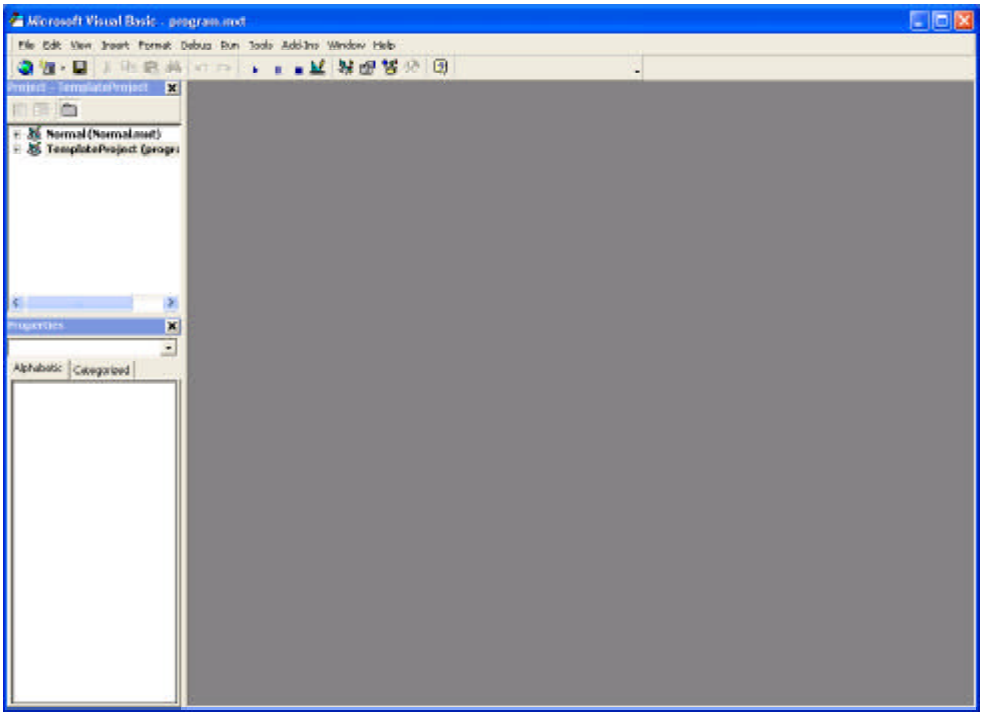


Figure 14-3 Visual Basic Editor Interface

✍ **7** **Click** at the radio button to the left of the label "A new empty map", and then **click** at the **OK** button to confirm the selection. This will close the dialog box.

✍ **8** **Click** at the **File** menu.

✍ **9** **Click** at the first name which appears under the Export Map... sub menu. This should be the name of the template file that was created above.

✍ **10** Click at the **Tools** menu and then at the **Macros** and **Visual Basic Editor** sub-menus, see Figure 14-3.

✍ **11** **Right-click** on the **TemplateProject** name to invoke the properties pop-up window, see Figure 14-4.

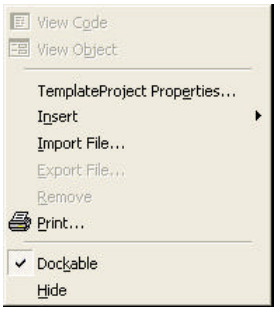


Figure 14-4 Properties Pop-Up

✍ **12** **Click** at the **TemplateProject Properties...** command. The Project Properties window should appear, see Figure 14-5.

✍ **13** **Click** on the **Protection** tab.

✍ **14** **Click** on the square to the left of the label "Lock project for viewing".

✍ **15** **Click** in the "Password" data field, and **enter** the password that is to be used to restrict access to the customizations provided in the template.

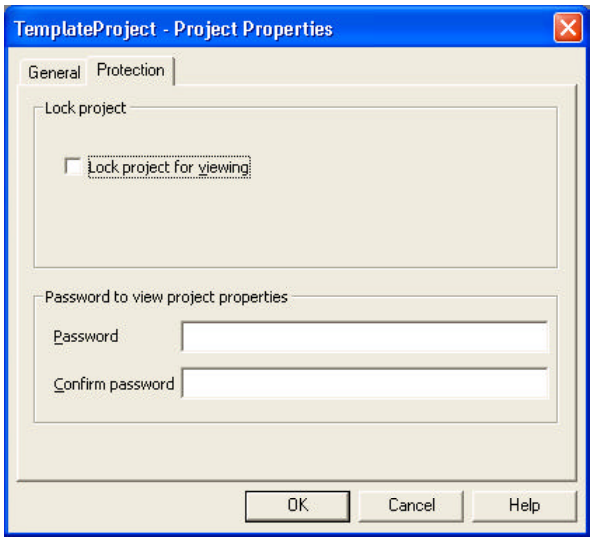


Figure 14-5 Template Protection Dialog Box

Passwords are case sensitive, that is, there is a difference between upper and lower case characters.

✍ **16** **Click** in the "Confirm password" data field, and **enter** the same password to confirm the password and then **click** at the **OK** button to complete the password protection phase.

✍ **17** **Click** at the **File** menu and then the **Close and Return to ArcMap** sub-menu.

✍ **18** **Click** at the **File** menu and then the **Save** sub-menu.

✍ **19** **Click** at the **File** menu and then the **Exit** sub-menu.

The template file has now been applied password protection and is ready to be referenced by an ArcMap™ document file.

14.2 Creating an ArcMap Project File referencing an ArcMap Template File

Once an ArcMap™ template file has been created, the developer has the option to either (a) distribute the template file with instructions on how the template file can be referenced, or (b) create an ArcMap™ document file which references the template file and then distribute both the ArcMap™ document file and template file. Note that the template file must be included with the document file. In using the second approach, the developer saves the end user the effort of creating a document file, which references the template file. A document file which references a template file will be considerably smaller in file size as compared to the template file size.

In distributing an ArcMap™ document file, consideration should be given to establishing a central distribution directory where all files that are associated with the application should reside.

The developer when distributing the application will want to give consideration to setting up a central distribution directory where the document and template files, along with any other data that needs to be distributed, should reside. The developer can then instruct the end user that the ArcMap™ document file can be initially opened from the central distribution directory, then using the Save As... command, a new document file can be created, which the end user can use to perform the necessary work.

✍ 1 **Invoke** the ArcMap™ program. The "Start using ArcMap with" selection box of Figure 14-2 is displayed.

✍ 2 **Click** at the radio button, which appears, to the left of the label "An existing map" of Figure 14-2, and then **click** at the **OK** button to confirm the selection. This will

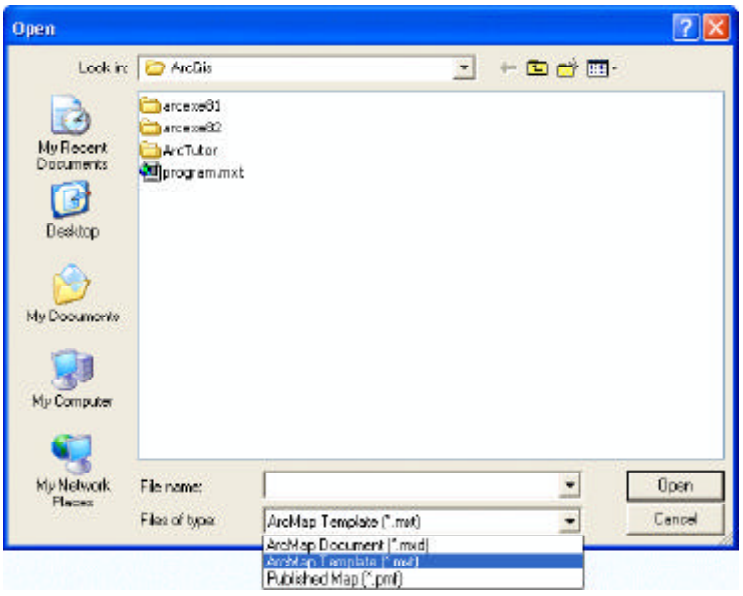


Figure 14-6 Open An Existing Map Dialog Box

close the dialog box, and will display the conventional file browsing window, similar to that of Figure 14-6

✍ **3** **Navigate** to the directory where the appropriate template file resides, **Scroll** down in the "Files of type:" combo box, and **click** at the ***.mxt** option. The template file should now appear.

✍ **4** **Click** at the name of the template file, in the file display area, and then **click** at the **Open** button to open it.

The typical ArcMap™ interface window is displayed. At this point we have created a new document file, which only has a reference to the template file. The document file has not been assigned a name.

✍ **5** **Click** at the **File** menu and then the **Save As...** sub-menu. The conventional file browsing window, similar to that of Figure 14-7 is now displayed.

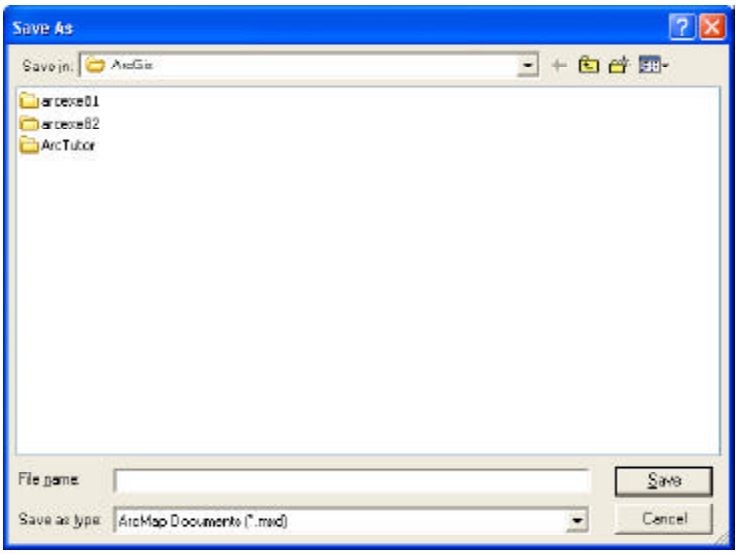


Figure 14-7 Save As Dialog Box

✍ **6** **Navigate** to the directory where the document file is to be stored, **click** in the "File name:" data field, and **enter** the desired document file name omitting any file name extension.

✍ **7** **Click** at the **Save** button to create the document file.

✍ **8** **Click** at the **File** menu and then at the **Exit** sub menu to exit ArcMap™.

In distributing an ArcMap™ document file, which references a template file, the template file must be included in the distribution.

The document file which has been created contains only a reference to the template file. The end user can begin working with this document file, adding data if need be, utilizing the customizations available in the template file. Since the template file is password protected, the customizations are safe from tampering.

14.3 Creating an Active X DLL

14.3.1 General Commentary

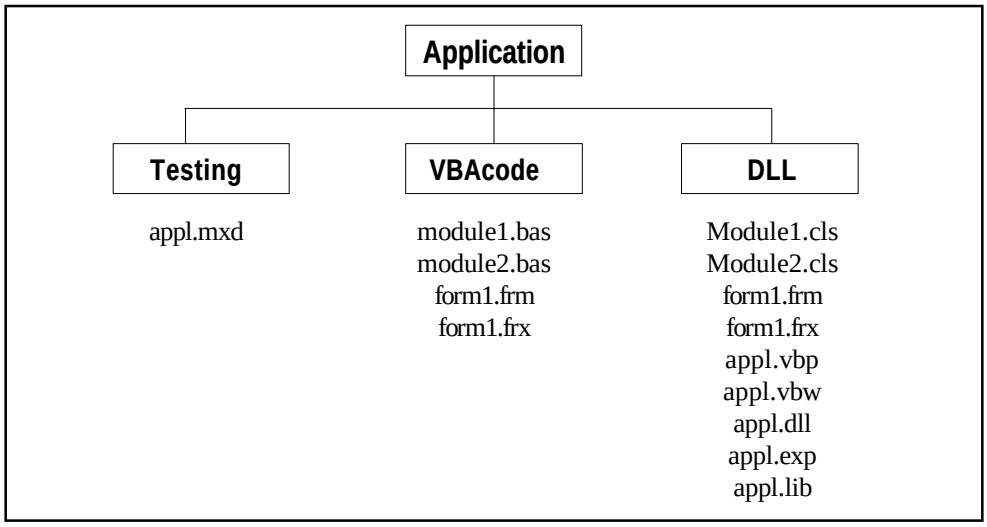
In the Avenue development environment, the developer had the ability to create extensions, which provided a means of distributing applications independent of ArcView project files. In the **VB**, Visual Basic, environment the developer has the ability to create Active X DLLs, which likewise, offers a means of distributing applications independent of ArcMap document files.

It is suggested that there should be one VB project file for every Active X DLL, which is created.

To create an Active X DLL, the developer creates Class modules within a Visual Basic workspace. A Visual Basic workspace is defined as a directory (folder) on disk where the Visual Basic project file is stored. Depending upon the type of functionality to be provided within the DLL, the structure of the Class modules will vary. For example, the structure of a Class module for a tool is different than that of a Combo box (drop-down). Depending upon the complexity of the DLL to be created, the Visual Basic project can contain one or several Class modules. That is to say, if the DLL is to deliver a single tool, the Visual Basic project file will contain a single Tool Class module. If the DLL is to deliver a toolbar with two tools, the Visual Basic project file will contain a Toolbar Class module, as well as, two Tool Class modules.

As stated in Section 1.3.1, prior to creating an Active X DLL, it is recommended that the developer perform the writing and debugging of the application in the **VBA** environment. Once the application is at a point for distribution, the Active X DLL(s) can be created. The code modules created in the **VBA** environment can be referenced in the Visual Basic workspace for incorporation into the Active X DLL. Note the use of the word "referenced" in the preceding statement. When an existing code module is added to a Visual Basic project, the actual source is not included in the Visual Basic project. That is to say, a Visual Basic project file simply contains pointers (pathnames) to the modules that comprise the project file. This is in stark difference to how the **VBA** environment operates. That is, all forms, code modules, etc. which comprise an application are stored within the ArcMap document file in which they were created.

Shown in Figure 14-8 is a possible directory structure for developing an ArcMap based application. As can be seen, a top level directory is created for the application. Within this directory, three sub-directories are created, **Testing**, **VBAcode** and **DLL**, although



A suggested directory structure for developing an ArcMap™ Active X DLL.

Figure 14-8 Sample Directory Structure for Creating Active X DLL

any naming convention can be used. These directories will contain certain information, as described below, which will be established during the development cycle.

The **Testing** directory is to contain the ArcMap document file, *appl.mxd*, where the application will be initially developed and debugged. As mentioned above, all of the code modules, forms, etc. are stored within the document file, just like an ArcView 3.x project file contained all of the dialogs and scripts that comprised an application. Therefore, for safety reasons, it is suggested that the **ExportVBAcode** Avenue Wrap be used to export the modules, forms, etc., within the ArcMap document file, into the **VBAcode** folder. In so doing, should "tragedy" strike the ArcMap document file, all would not be lost.

Since the code modules are stored externally from the Visual Basic project file, it is possible to use any text editor to modify the files. When the Visual Basic project file is reopened, any modifications made to the code modules will be visible.

The **VBAcode** directory, as stated above, provides a repository for the code modules forms, etc., external to the ArcMap document file. In addition to providing this service, it is the code modules in this directory that will be added to the Visual Basic project file, see Step 20, Section 14.3.2.

The **DLL** directory is to contain the Visual Basic project file, *appl.vbp*, which builds the Active X DLL for the application. In addition, all of the Class modules and forms comprising the Active X DLL will also be stored in this directory. Note that the forms created, in the **VBA** environment within the ArcMap document file, will need to be recreated in the **VB** environment. The code modules in the **VBAcode** directory will be added to the Visual Basic project file. So that, the **DLL** directory will contain only those modules specific to building the Active X DLL.

14.3.2 Creating a Toolbar DLL

This section provides information describing how a DLL can be created such that a single toolbar with one or more tools on the toolbar is provided. Note that a tool is different from a ComboBox, in that, there are different "events" and "properties" that are associated with a tool and not with a ComboBox, although there are some commonalities. Section 14.3.4 discusses how a ComboBox can be created.

✍ **1** Using Windows Explorer **create the DLL directory** (folder) as shown in Figure 14-8.

✍ **2** **Invoke** Visual Basic Version 6.0.

✍ **3** **Select** the **New** tab, followed by **clicking** on the **Active X DLL** icon and then **select** the **Open** button, see Figure 14-9.

Figure 14-9 VB 6.0 Splash Box

✍ **4** At this point an empty Class module will be established. The developer can begin to enter the appropriate code for the type of Class module to be created.

Establish the Toolbar Class Module. This module contains the code which defines what tools are to appear on the Toolbar. Note, it is possible for a toolbar to contain Tool and ComboBox controls. A Toolbar does not have to contain only one type of control.

In this discussion, we will insert code for a Toolbar. Shown in Figure 14-10 is "boiler-plate" or "stub" code for a Toolbar Class module. The developer can paste this code into the Class module window and modify the code as desired. Comments within the "stub" code describe what information is specified in a particular procedure.

Once the code has been modified, using the Properties window for the Class module set the Instancing and Name properties of the Class module, see Figure 14-11. Note any appropriate name can be entered in the name property data field. The Instancing property should be set to GlobalMultiUse.

```
'
' * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
' *                                                                       *
' *  Name: ToolBar                          File Name: ToolBar.cls       *
' *                                                                       *
' * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
' *                                                                       *
' *  PURPOSE:  CLASS MODULE DEFINING THE TOOLBAR TO BE DISTRIBUTED        *
' *            IN THE FORM OF AN ACTIVE X DLL                             *
' *                                                                       *
' *  GIVEN:    nothing                                                    *
' *                                                                       *
' *  RETURN:   nothing                                                    *
' *                                                                       *
' * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
'
Implements IToolBarDef

Private Property Get IToolBarDef_ItemCount() As Long
'
'   ---Define the number of items on the toolbar, in this case there
'   ---will be two commands (tools) which will appear on the toolbar
    IToolBarDef_ItemCount = 2
'
End Property

Private Sub IToolBarDef_GetItemInfo(ByVal pos As Long, _
                                    ByVal itemDef As esriCore.IItemDef)
'
'   ---Add the commands to the toolbar, noting that Project1 denotes
'   ---the name of the VB project being worked on.  A period is used
'   ---to separate the VB project name from the name of the class
'   ---module containing the command that is to appear on the toolbar
'   ---The programmer can change the name of the VB project and
'   ---command class module in the VB Properties window
  Select Case pos
  Case 0
    itemDef.ID = "Project1.Command1"
  Case 1
    itemDef.ID = "Project1.Command2"
  End Select
'
End Sub

Private Property Get IToolBarDef_Name() As String
'
   IToolBarDef_Name = "Custom Toolbar"
'
End Property

Private Property Get IToolBarDef_Caption() As String
'
'   ---Define the name of the toolbar which will appear in the
'   ---ArcMap Customize dialog box, the end user can toggle the
'   ---display of the toolbar by clicking in the square to the
'   ---left of this name
    IToolBarDef_Caption = "Custom Toolbar"
'
End Property
```

Figure 14-10 "Stub" code for a Toolbar Class Module

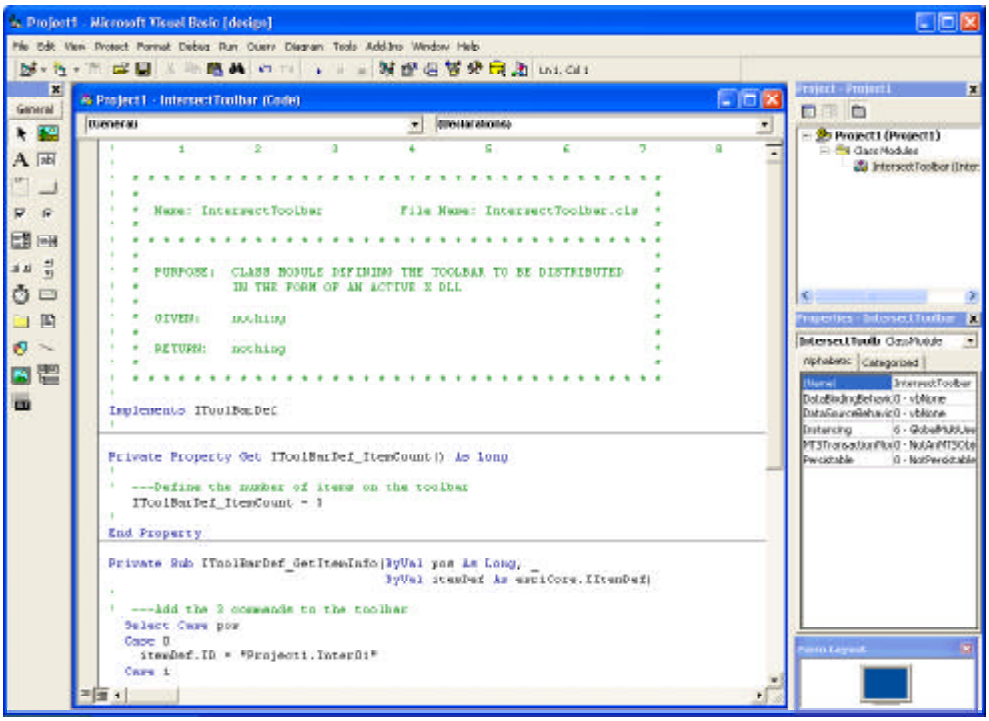


Figure 14-11 Class Module Properties Window

Establish the Tool Class Module. This module contains the code which defines how a tool operates, such as, the pop-up help message, the procedure that gets executed when the mouse button is depressed and so forth.

✍ **5** At this point we need to create a Class module for every tool that is to be added to the toolbar. The **Project** menu **Add Class Module** sub menu item can be used to create a new Class module, see Figure 14-12(a).

Figure 14-12(a) Creating a New Class Module

✍ **6** **Select** the **New** tab, followed by **clicking** on the **Class Module** icon and then **select** the **Open** button, see Figure 14-12(b). A new empty Class module window will appear.

Insert code for a Tool. Shown in Figure 14-13 is "boiler-plate" or "stub" code for a Tool Class module. The developer can paste this code into the Class module window and modify the code as desired. Comments within the "stub" code describe what information is specified in a particular procedure.

Once the code has been modified, using the Properties window for the Class module set the Instancing and Name properties of the Class module, see Figure 14-11.

✍ **7** Repeat Steps 5 and 6 for every tool to appear on the toolbar.

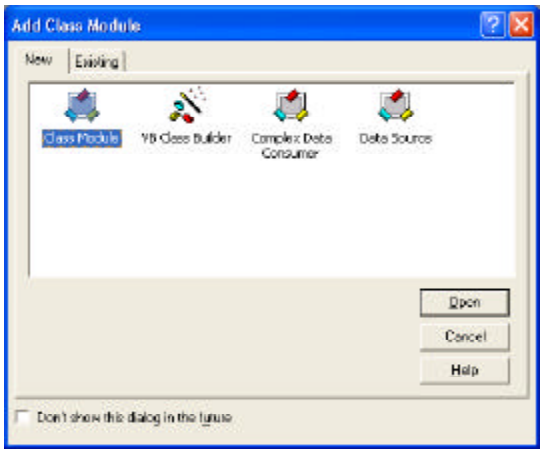


Figure 14-12(b)
Creating a New Class Module

At this point, the Class modules for the toolbar and the tools to appear on the toolbar have been created.

```
Option Explicit
'
' * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
' *                                                                       *
' *  Name: Command1                          File Name: Command1.cls      *
' *                                                                       *
' * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
' *                                                                       *
' *  PURPOSE:  CLASS MODULE DEFINING A COMMAND WHICH WILL BE ADDED        *
' *            TO A TOOLBAR IN THE FORM OF A TOOL                         *
' *                                                                       *
' *  GIVEN:    nothing                                                    *
' *                                                                       *
' *  RETURN:   nothing                                                    *
' *                                                                       *
' * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
'
Private m_pApp As IApplication
Private m_pDoc As IMxDocument
Private m_pMap As IMap
Private m_pExt As IExtensionConfig
'
Implements ITool
Implements ICommand
'

Private Sub Class_Terminate()
'
'   ---Clear member variables
   Set m_pMap = Nothing
   Set m_pDoc = Nothing
   Set m_pExt = Nothing
   Set m_pApp = Nothing
'
End Sub
```

Figure 14-13 "Stub" code for a Tool Class Module

```
Private Property Get ICommand_Enabled() As Boolean
'
'   ---Command is enabled only if there is data in the map
    If m_pMap.LayerCount > 0 Then
       ICommand_Enabled = True
    Else
       ICommand_Enabled = False
    End If
'
End Property

Private Property Get ICommand_Checked() As Boolean
'
    ICommand_Checked = False
'
End Property

Private Property Get ICommand_Name() As String
'
'   ---Set the internal name of this command. By convention, this
'   ---name string contains the category and caption of the command
    ICommand_Name = "CEDRA_AVcad_Tools.Point01"
'
End Property

Private Property Get ICommand_Caption() As String
'
'   ---Set the string that appears when the command is used as a
'   ---menu item. This name appears in the Commands window on the
'   ---right side of the Commands tab in the Customize dialog box
    ICommand_Caption = "Point01"
'
End Property

Private Property Get ICommand_Tooltip() As String
'
'   ---Define the pop-up help
    ICommand_Tooltip = "Define points by picking"
'
End Property

Private Property Get ICommand_Message() As String
'
'   ---Set the message string that appears in the status bar of the
'   ---application when the mouse passes over the command
    ICommand_Message = "Make a pick"
'
End Property

Private Property Get ICommand_HelpFile() As String
'
'
End Property

Private Property Get ICommand_HelpContextID() As Long
'
'
End Property
```

Figure 14-13 "Stub" code for a Tool Class Module (continued)

```
Private Property Get ICommand_Bitmap() As esriCore.OLE_HANDLE
'
'   ---Get command bitmap from the image list on the form call Form1
    ICommand_Bitmap = Form1.ImageList1.ListImages(1).Picture
'
End Property

Private Property Get ICommand_Category() As String
'
'   ---Set the category of this command. This determines where the
'   ---command appears in the Commands panel of the Customize dialog
    ICommand_Category = "CEDRA_AVcad_Tools"
'
End Property

Private Sub ICommand_OnCreate(ByVal hook As Object)
'
'   ---The hook argument is a pointer to Application object,
'   ---establish a hook to the application
    Set m_pApp = hook
    Set m_pDoc = m_pApp.Document
    Set m_pMap = m_pDoc.FocusMap
'
'   ---Transfer the IApplication object into global memory
    Set ugm_pApp = m_pApp
'
'   ---Initialize the CEDRA Avenue Wraps global variables
    Call avInit(m_pApp)
'
End Sub

Private Sub ICommand_OnClick()
'
'   ---Add code to do some action when the command is clicked
'
End Sub

Private Property Get ITool_Cursor() As esriCore.OLE_HANDLE
'
'   ---Set the cursor of the command
'
End Property

Private Sub ITool_OnMouseDown(ByVal button As Long, _
                ByVal shift As Long, ByVal X As Long, ByVal Y As Long)
'
'   ---Add code to do some action when a mouse button is pressed
'
End Sub

Private Sub ITool_OnMouseMove(ByVal button As Long, _
                ByVal shift As Long, ByVal X As Long, ByVal Y As Long)
'
'   ---Add code to do some action when a mouse is moved
'
End Sub

Private Sub ITool_OnMouseUp(ByVal button As Long, _
                ByVal shift As Long, ByVal X As Long, ByVal Y As Long)
'
```

Initializing the Avenue Wraps DLL, avwraps.dll, globals in a VB project file.

The OnMouse events have as one of their arguments the parameter ***button*** which reflects the mouse button that was used. The values that ***button*** can be assigned include: **0** No button, **1** Left button is pressed, **2** Right button is pressed and **4** Middle button is pressed.

```
'   ---Add code to do some action when a mouse button is released
'
End Sub

Private Sub ITool_OnDblClick()
'
'   ---Add code to do some action on double-click
'
End Sub

Private Sub ITool_OnKeyDown(ByVal keyCode As Long, _
                            ByVal shift As Long)
'
'   ---Add code to do some action when a keyboard button is pressed
'
End Sub

Private Sub ITool_OnKeyUp(ByVal keyCode As Long, ByVal shift As Long)
'
'   ---Add code to do some action when a keyboard button is released
'
End Sub

Private Function ITool_OnContextMenu(ByVal X As Long, _
                                     ByVal Y As Long) As Boolean
'
'   ---Add code to show custom context menu when there is a
'   ---right click
'
End Function

Private Sub ITool_Refresh(ByVal hDC As esriCore.OLE_HANDLE)
'
'   ---Add code to do something when the screen display in the
'   ---application is refreshed
'
End Sub

Private Function ITool_Deactivate() As Boolean
'
'   ---Deactivate the tool. If ITool_Deactivate is not set to be
'   ---true no other tool on the tool bar will be able to be selected
'
'   ---Handle any errors that may occur
    On Error GoTo Errorhandler
'
    ITool_Deactivate = True
'
    Exit Function
'
'   ---Handle any errors that were detected
Errorhandler:
'
'   ---Display the detected error
    MsgBox "Error " & Err.Number & " - " & Err.Description & _
           Chr(13) & "Function: ITool_Deactivate"
'
End Function
```

Figure 14-13 "Stub" code for a Tool Class Module (continued)

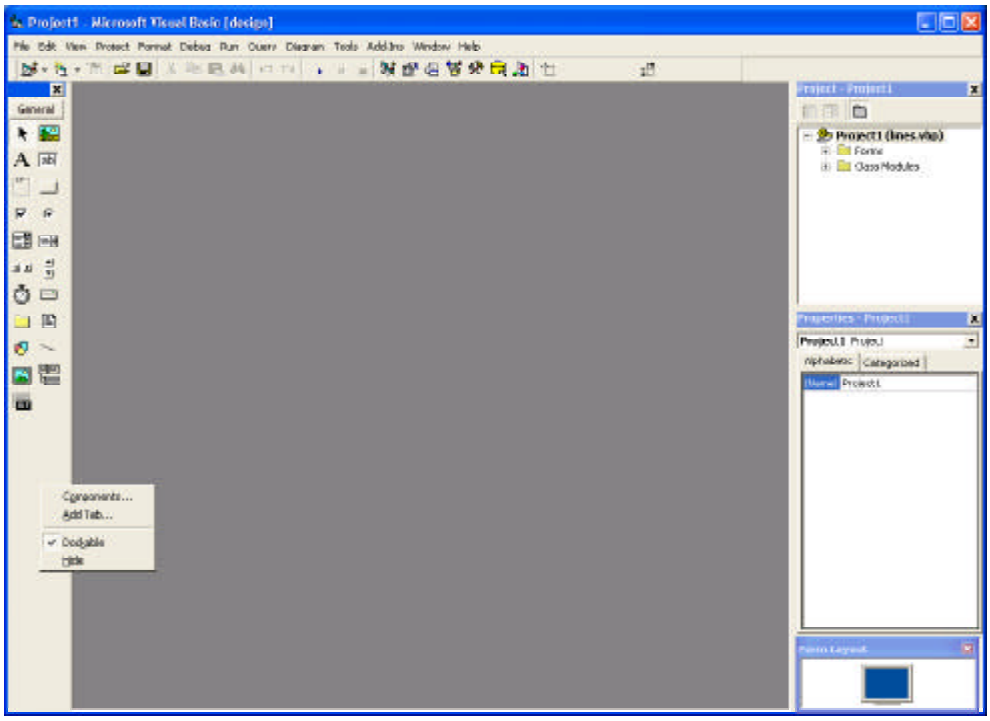


Figure 14-14(a) Components "Pop-Up" Window Menu Display

✍ **8** Move the cursor into a blank area within the toolbar display on the left side of the VB 6.0 application window and **right-click**. A pop-up menu will appear, see Figure 14-14(a). **Click** on the **Components...** menu item, which appears in the pop-up menu list.

✍ **9** **Click** on the square to the left of the label "Microsoft Windows Common Controls 6.0", see Figure 14-14(b).

✍ **10** **Click** at the **OK** button to load the additional controls. The ImageList tool will be one of the additional tools added to the project file.

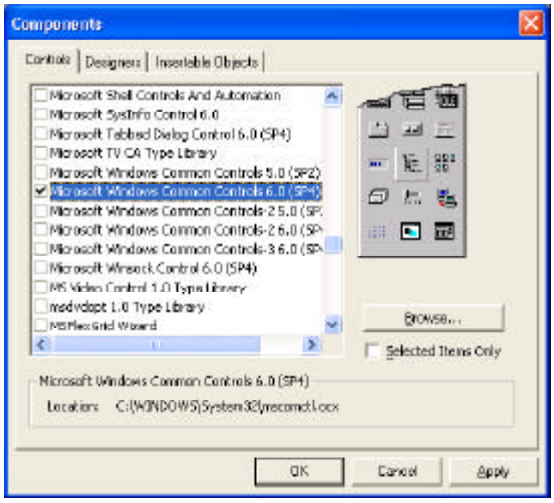


Figure 14-14(b)
Available Components

Every tool on the toolbar can have an icon or bitmap image assigned to it. To do so, additional commands need to be added to the Visual Basic project file. In this case an ImageList control can be used to assign a tool a specific icon. The ImageList control is available in the Microsoft Windows Common Controls compo-

✍ **11** **Select** the **Project** menu and **Add Form** sub menu item.

✍ **12** **Select** the **New** tab, followed by **clicking** on the **Form** icon and then **select** the **Open** button, see Figure 14-15.

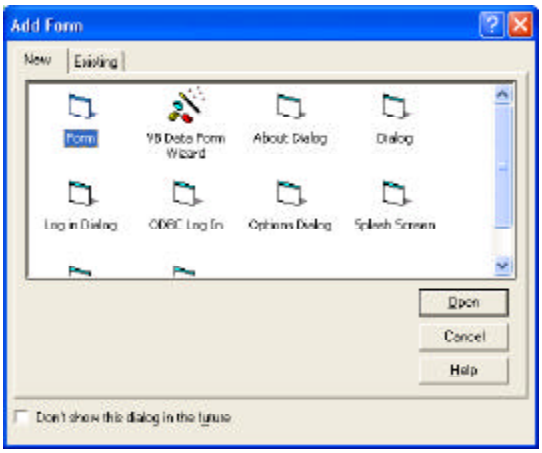


Figure 14-15
Available Form Types

✍ **13** **Select** the ImageList tool and drag a rectangle within the form to add an ImageList control to the form.

✍ **14** **Right-click** on the ImageList control that was just added and **select** the **Properties** menu from the pop-up menu list as shown Figure 14-16.

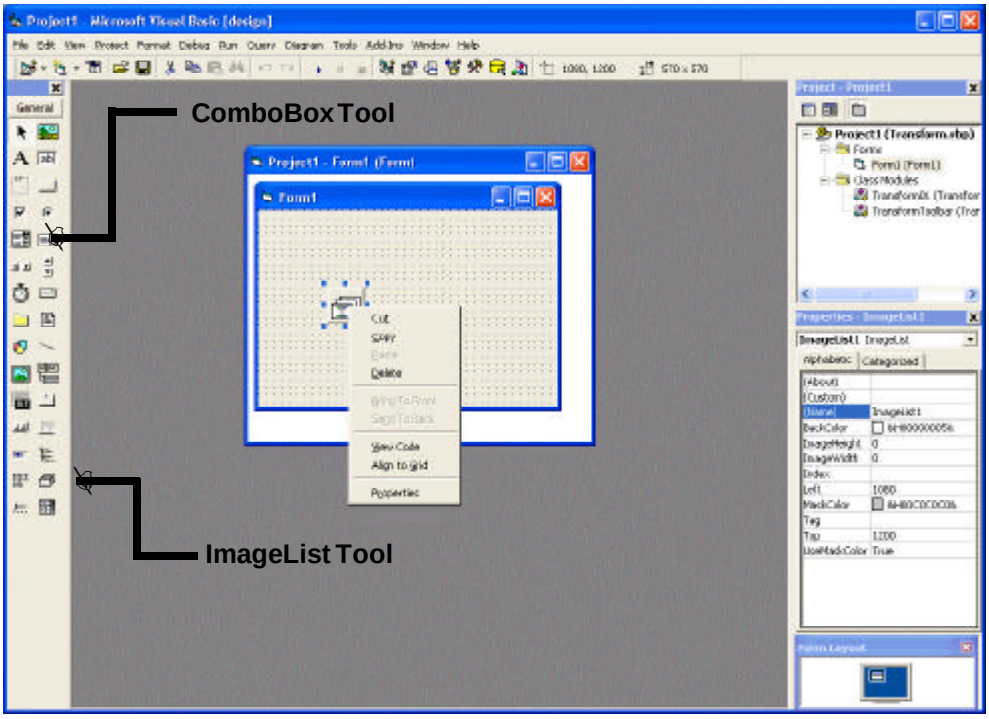


Figure 14-16 ImageList Control Pop-Up Menu Options

✍ **15** **Select** the **Images** tab, followed by **clicking** on the **Insert Picture** button, see Figure 14-17, navigate to the directory which contains the desired bitmap (icon), which is to be assigned to the tool and then **select** the **Open** button.

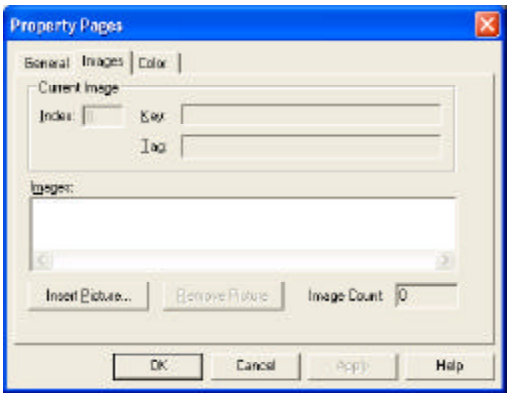


Figure 14-17
Available Components

✍ **16** Repeat Step 15 for every tool to appear on the toolbar. When all icons have been added, **select** the **OK** button. The order in which the icons appear in Figure 14-17 correspond to the order of the tools on the toolbar, left to right.

✍ **17** Select the **Project** menu and the **Add Module** sub menu item.

✍ **18** **Select** the **New** tab, followed by **clicking** on the **Module** icon and then **select** the **Open** button, an empty module window should appear.

✍ **19** Enter the following statement in the empty window which was created by the previous step.

```
Public ugm_pApp As IApplication
```

At this point a global variable called ugm_pApp is being established which will be used to transfer the IApplication interface from the Tool Class module to the rest of the code modules comprising the application. In the ICommand_OnCreate procedure, the ugm_pApp object is established and since is it declared as Public, it is available to all of the modules in the application. Note that the ICommand_OnCreate procedure is also used to initialize the Avenue Wraps global variables, see Figure 14-13.

In the VBA environment, the Application object is available and is typically used to get access to the IApplication interface. In the VB environment, the Application object is not available. To get around this, the OnCreate method is employed to get a hook to the IApplication interface and the Public ugm_pApp object used to pass the interface to the other modules in the application.

✍ **20** Select the **Project** menu and the **References...** sub menu item.

✍ **21** **Click** on the squares to the left of the labels "ESRI ArcMap Object Library" and "ESRI Object Library" and then **select** the **OK** button, see Figure 14-18.

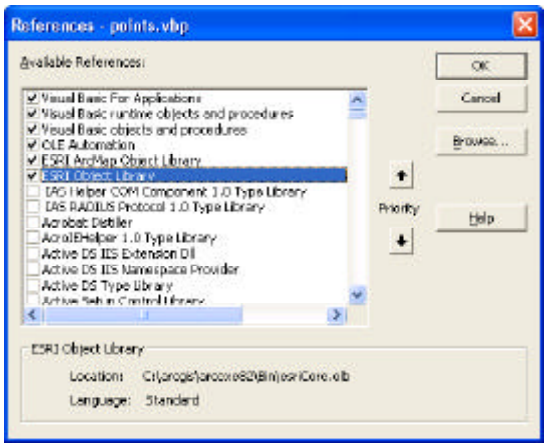


Figure 14-18
Available COM Components

✍ **22** Select the **File** menu and the **Save Project As...** sub menu item, navigate to the directory created in Step 1 and **click** the **Save** button, see Figure 14-19. The Save button will need to be clicked for every form, command module, toolbar Class module that was created, as well as, for the VB project file. As mentioned in Section 14.3.1 all forms, modules, etc. are stored external to the VB project file, so that, when the project file is first saved all of the forms, modules, etc. have to be given a name so that they can be written to disk. After the initial saving, any modifications which may be made can be saved with the Save Project sub menu item, without having to specify a name for each of the forms, modules, etc.

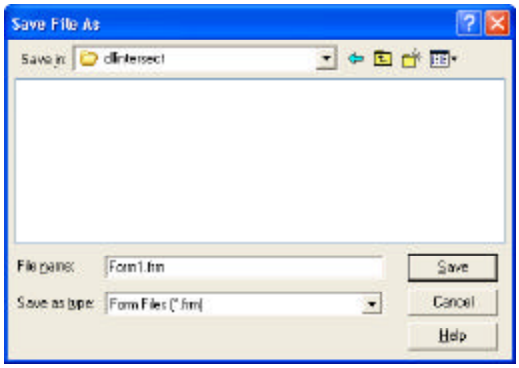


Figure 14-19
Save As Dialog Box

✍ **23** Add any other modules, which may be needed by the application, with the **Project** menu and **Add Module** sub menu item. These should be the modules which reside in the VBAcode directory, described in Section 14.3.1.

✍ **24** **Select** the **File** menu and the **Make x.dll** sub menu item, where x is the name of the VB project file, **enter** the **name of the .dll file** to be created and **click**

the **OK** button to create the Active X DLL, see Figure 14-20. Note that the name of the .dll file does not have to be the same as the name of the VB project file.

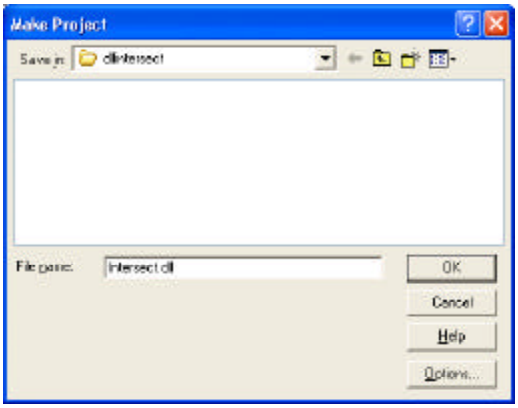


Figure 14-20
Make DLL Dialog Box

At this point, Visual Basic will compile the project checking that all referenced procedures have been found and if any compilation errors were detected. If an error is detected, the developer is informed accordingly. At this point, the error must be resolved prior to continuing with the compilation. Once the error has been resolved, repeat this step until all errors are eliminated. When no more errors are detected, the Active X DLL file is established. At this point, the .dll file can be added within ArcMap and its functionality made available to the end user.

14.3.3 Loading an Active X DLL within ArcMap

Once an Active X DLL has been created, it can be loaded within ArcMap and its functionality employed. To accomplish this perform the following:

✍ **1** **Invoke** ArcMap, at which time the default splash box is displayed. Accept the default selection to create a new empty map, and **click** at the **OK** button.

✍ **2** **Click** at the **Tools** menu and then at the **Customize...** sub-menu item. **Select** the **Add from File...** button, see Figure 14-21.

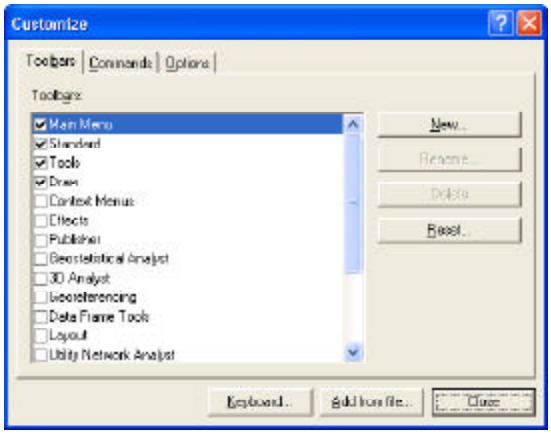


Figure 14-21
Customize Dialog Box

✍ **3** **Navigate** to the directory containing the

Active X DLL to be loaded, **click** on the **name of the .dll file** and **select** the **Open** button.

✍ **4** Wait until the Added Objects... box appears, see Figure 14-22 for a typical message box display. Depending upon the size the .dll file this may take a few seconds. Once the message box appears, **click** the **OK** button.

Figure 14-22
COM Objects Added Message Box

✍ **5** On the left side of the dialog box, as shown in Figure 14-21, **scroll** down until the name of the toolbar, which is included in the .dll file, appears. **Click** on the square to the left of the name to make the toolbar visible.

✍ **6** **Click** the **Close** button to dismiss the Customize dialog box.

The toolbar should now be visible and its functionality available to the end user.

14.3.4 Creating a ComboBox

Creating a ComboBox control is similar to creating a Tool control with the following exceptions:

1. The "stub" code for a ComboBox control is different from a Tool control. In Figure 14-23 the "stub" code for a ComboBox control is shown. So that in Step 6 of Section 14.3.2 paste the "stub" code shown in Figure 14-23 in the Class module.

2. Rather than adding an ImageList control to the form and loading icons, as done in Steps 13 through 16 of Section 14.3.2, a ComboBox control will be added to the form and code for the control's Click event and loading of the form will be written. Shown in Figure 14-24 is the code for the control's Click event and the Form_Load procedure. So that, replace Steps 13 through 16 of Section 14.3.2 with the following:

   ✍ **13** **Select** the ComboBox tool, see Figure 14-16, and drag a rectangle within the form to add a ComboBox control to the form.

```
Option Explicit
'
' * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
' *                                                                       *
' *  Name: ComboBox1                        File Name: ComboBox1.cls      *
' *                                                                       *
' * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
' *                                                                       *
' *  PURPOSE:  CLASS MODULE DEFINING A COMBOBOX COMMAND WHICH CAN         *
' *            BE ADDED TO A TOOLBAR OR DRAGGED INDIVIDUALLY ONTO         *
' *            THE ARCMAP USER INTERFACE                                  *
' *                                                                       *
' *  GIVEN:    nothing                                                    *
' *                                                                       *
' *  RETURN:   nothing                                                    *
' *                                                                       *
' * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
'
Private m_pApp As IApplication
Private m_pDoc As IMxDocument
Private m_pMap As IMap
Private m_pExt As IExtensionConfig
'
Implements ICommand
Implements IToolControl
'

Private Sub Class_Terminate()
'
'  ---Clear member variables
   Set m_pMap = Nothing
   Set m_pDoc = Nothing
   Set m_pExt = Nothing
   Set m_pApp = Nothing
'
End Sub

Private Property Get ICommand_Enabled() As Boolean
'
'  ---Command is enabled only if there is data in the map
   If m_pMap.LayerCount > 0 Then
      ICommand_Enabled = True
   Else
      ICommand_Enabled = False
   End If
'
End Property

Private Property Get ICommand_Checked() As Boolean
'
   ICommand_Checked = False
'
End Property

Private Property Get ICommand_Name() As String
'
'  ---Set the internal name of this command. By convention, this
'  ---name string contains the category and caption of the command
   ICommand_Name = "CEDRA_AVcad_Menus.Annotation_Commands"
'
End Property
```

Figure 14-23 "Stub" code for a ComboBox Class Module

```
Private Property Get ICommand_Caption() As String
'
'   ---Set the string that appears when the command is used as a
'   ---menu item. This name appears in the Commands window on the
'   ---right side of the Commands tab in the Customize dialog box
    ICommand_Caption = "Annotation_Commands"
'
End Property

Private Property Get ICommand_Tooltip() As String
'
'   ---Define the pop-up help
    ICommand_Tooltip = "Annotation commands"
'
End Property

Private Property Get ICommand_Message() As String
'
'   ---Set the message string that appears in the status bar of the
'   ---application when the mouse passes over the command
    ICommand_Message = " "
'
End Property

Private Property Get ICommand_HelpFile() As String
'
'
End Property

Private Property Get ICommand_HelpContextID() As Long
'
'
End Property

Private Property Get ICommand_Bitmap() As esriCore.OLE_HANDLE
'
'
End Property

Private Property Get ICommand_Category() As String
'
'   ---Set the category of this command. This determines where the
'   ---command appears in the Commands panel of the Customize dialog
    ICommand_Category = "CEDRA_AVcad_Menus"
'
End Property

Private Sub ICommand_OnCreate(ByVal hook As Object)
'
'   ---The hook argument is a pointer to Application object,
'   ---establish a hook to the application
    Set m_pApp = hook
    Set m_pDoc = m_pApp.Document
    Set m_pMap = m_pDoc.FocusMap
'
'   ---Transfer the IApplication object into global memory
    Set ugm_pApp = m_pApp
'
'   ---Initialize the CEDRA Avenue Wraps global variables
    Call avInit(m_pApp)
'
End Sub
```

Initializing the Avenue Wraps DLL, avwraps.dll, globals in a VB project file.

Figure 14-23 "Stub" code for a ComboBox Class Module (continued)

```
Private Sub ICommand_OnClick()
'
'   ---Add code to do some action when the command is clicked
'
End Sub

Private Property Get IToolControl_hWnd() As esriCore.OLE_HANDLE
'
'   ---Form1 denotes the name of the form and Combo1 denotes the
'   ---name of the ComboBox control
    IToolControl_hWnd = Form1.Combo1.hWnd
'
End Property

Private Function IToolControl_OnDrop(ByVal barType As _
                                esriCore.esriCmdBarType) As Boolean
'
    If barType = esriCmdBarTypeToolbar Then
       IToolControl_OnDrop = True
    End If
'
End Function

Private Sub IToolControl_OnFocus(ByVal complete As _
                                esriCore.ICompletionNotify)
'
'   ---The SetComplete method on the ICompletionNotify object
'   ---is called in the combobox click event
    Set pCompNotify = complete
'
End Sub
```

Figure 14-23 "Stub" code for a ComboBox Class Module (continued)

✍ **14** **Double-click** in a blank area within the form. The code module associated with the form will appear.

✍ **15** **Paste** the code shown in Figure 14-24 in the code module window and make the appropriate modifications. Specifically, the options that should appear in the ComboBox's drop-down list and which procedures are to be executed when an option is selected.

✍ **16** **Set** the Name property of the Form using the appropriate data field in the Properties window. Note this name will be referenced in the IToolControl_hWnd procedure of Figure 14-23.

3. Replace the code that was entered in Step 19 of Section 14.3.2 with the following code:

```
Public ugm_pApp As IApplication
Public pCompNotify As ICompletionNotify
```

```
Option Explicit
'
' * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
' *                                                                       *
' *  Name: Form1                                File Name: Form1.frm      *
' *                                                                       *
' * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
' *                                                                       *
' *  PURPOSE:  Populate and execute commands in a ComboBox control        *
' *                                                                       *
' *  GIVEN:    nothing                                                    *
' *                                                                       *
' *  RETURN:   nothing                                                    *
' *                                                                       *
' * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
'
'

Private Sub Combo1_Change()
'
'
End Sub

Private Sub Combo1_Click()
'
'   ---Let the application know that the combobox control
'   ---no longer needs focus after an item is selected in
'   ---the combobox
   pCompNotify.SetComplete
'
'   ---Execute the command associated with the item that was
'   ---selected in the combo box
   Select Case Combo1.ListIndex
   Case 0
      Call AnnDistance
   Case 1
      Call AnnAzimuth
   End Select
'
End Sub

Private Sub Form_Load()
'
'   ---Delete all items in the combo box
   Combo1.Clear
'
'   ---Add the desired items to the combo box
   Combo1.AddItem "Annotate Distance"
   Combo1.AddItem "Annotate Azimuth"
'
'   ---Set the default item for the combo box
   Combo1.Text = "Annotate Distance"
'
End Sub
```

Figure 14-24 ComboBox Control Code

The ComboBox control requires another Public object, pCompNotify, which is why we have an additional Public object defined in the code module.

4. At this point the ComboBox control has been defined.

14.4 Using the Avenue Wraps DLL

14.4.1 General Commentary

In the Avenue Wraps distribution directory there will be a folder called DLL, which contains the DLL file, *avwraps.dll*. This is the **D**ynamically **L**inked **L**ibrary file containing all of the Avenue Wraps discussed in the previous sections. The file *avwraps.dll* can be referenced in the VBE environment within ArcMap or in a VB project file. Using the *avwraps.dll* file enables the developer to create an application without having the source for the Avenue Wraps included in the application, although the *avwraps.dll* file will need to be included when distributing the application.

To minimize the size of an application, reference the Avenue Wraps DLL, avwraps.dll, rather than including the source in the application.

To reference the Avenue Wraps DLL in the VBE environment within ArcMap:

✍ **1** **Invoke** ArcMap, **click** the radial button to the left of the *A new empty Map* label and select the **OK** button.

✍ **2** **Click** at the **Tools** menu and then at the **Macros** and **Visual Basic Editor** sub-menus to display the VBE work environment of Figure 14-3.

Referencing the Avenue Wraps DLL, avwraps.dll, in VBA.

✍ **3** **Click** at the **Tools** menu and then at the **References...** sub-menu.

✍ **4a** If the *avwraps.dll* file is to be referenced for the very first time:

select the **Browse...** button, **navigate** to the directory where the *avwraps.dll* file resides, **select** the file, **click** the **Open** button and then **click** the **OK** button. Proceed to Step 5.

✍ **4b** If the *avwraps.dll* file has previously been referenced on the computer:

scroll down in the list on the left side of the dialog box and **click** in the square to the left of the *CEDRA Avenue Wraps* label and **click** the **OK** button.

✍ **5** **Click** in the square containing the plus (+) character to the left of the folder called ArcMap Objects under the Project group in the Project window.

Initializing the Avenue Wraps DLL, avwraps.dll, globals in VBA.

✍ **6** **Double-click** on the **ThisDocument** module name.

✍ **7** **Scroll** down in the Object drop-down list and **select** the **MxDocument** name.

✍ **8** **Scroll** down in the Procedure drop-down list and **select** the **OpenDocument** name.

✍ **9** **Insert** the line **Call avInit(Application)** in the OpenDocument procedure.

✍ **10** **Click** the **Run Sub/UserForm tool** to execute the subroutine. This will initialize the Avenue Wraps global variables.

The *avwraps.dll* has now been referenced in the VBA application, and all of the Avenue Wraps are now available to the developer. The user can now create new modules and begin to convert existing Avenue code or develop new code using the Avenue Wraps "wraparounds".

Note that any time a new module is inserted in the ArcMap document file, the OpenDocument procedure will need to be re-executed. The OpenDocument procedure is a good location to perform any initialization that may be required.

Referencing the Avenue Wraps DLL, avwraps.dll, in VB.

To reference the Avenue Wraps DLL in a VB project file:

✍ **1** **Invoke** Visual Basic Version 6.0.

✍ **2** **Select** the **New** tab, followed by **clicking** on the **Active X DLL** icon and then **select** the **Open** button.

✍ **3** **Click** at the **Project** menu and then at the **References...** sub-menu.

✍ **4a** If the *avwraps.dll* file is to be referenced for the very first time:

select the **Browse...** button, **navigate** to the directory where the *avwraps.dll* file resides, **select** the file, **click** the **Open** button and then **click** the **OK** button.

✍ **4b** If the *avwraps.dll* file has previously been referenced on the computer:

scroll down in the list on the left side of the dialog box and **click** in the square to the left of the *CEDRA Avenue Wraps* label and **click** the **OK** button.

The Avenue Wraps DLL has now been referenced and all of its procedures are available to the developer for use. Note that the Avenue Wraps global variables will need to be initialized. This is typically done in the ICommand_OnCreate procedure, see Figure 14-13.

In helping to get started, the folder DATA, within the Avenue Wraps distribution directory, contains an ArcMap document file, *exercise.mxd*, that references the Avenue Wraps dll, *avwraps.dll*. As mentioned above, the ArcMap document file does not have to contain the Avenue Wraps source modules when the Avenue Wraps dll file is referenced in the document file. Within this document file there are sample VBA macros demonstrating various Avenue Wraps functionality. Each module can be opened and executed using the VBA Run tool. In additon, there is a custom tool bar that demonstrates various types of user interaction with the map display, such as, making a single pick, drawing a line, drawing a circle, etc. The source code for the tool bar commands are in the ThisDocument module under the ArcMap Objects folder within the Project category. Holding the cursor over a specific tool will result in a ToolTip message being displayed. As such, the document file, *exercise.mxd*, serves as an excellant starting point for the developer who wishes to begin converting an Avenue based application into the ArcGIS environment. The developer should feel free to modify this document file as desired.

In Getting Started, the *exercise.mxd* file, located in the Avenue Wraps distribution directory, can be used as a beginning point. Note that this document file was built using Version 8.2 of ArcGIS.

14.4.2 Avenue Wraps Properties

When using the Avenue Wraps DLL file, *avwraps.dll*, the developer may wish to access certain of the Public variables which are used in the DLL file. In this situation, it is not possible to address the Public variable by its specific name but rather, the Public variable needs to be addressed by its Property name. For example, the Public variable *ugEditOp* can be accessed by using *avwraps.EditOp*, not ugEditOp. Likewise, the Public variable *ugcwDirName* can be accessed by using *avwraps.WorkDir*. As can be seen the convention is to use *avwraps.* followed by the *Property name* for the particular Public variable. This convention needs to be followed when using the Avenue Wraps DLL, *avwraps.dll*, in an application. If the Avenue Wraps source is included in the application, then the Public variable name can be used as is and the convention mentioned above ignored.

Shown below is a table of the Avenue Wraps Public variables and their corresponding Property names.

Public Variable Name	Property	Description
ugAppName	AppName	Internal application name
ugcwDirName	WorkDir	Current working directory
ugEditMode	EditMode	Editor mode or status
ugEditOp	EditOp	Editor operation for Undo
ugerror	Error	Error detection flag
ugFalseX	FalseX	Spatial reference false X

Public Variable Name	Property	Description
ugFalseY	FalseY	Spatial reference false Y
ugfNamelastchar	PathCharacter	Path name delineator: \
ughDialogBoxData	HDialogData	Horizontal Dialog contents
ughDialogBoxLeft	HDialogLeft	Horizontal Dialog X position
ughDialogBoxRight	HDialogRight	Horizontal Dialog Y position
ugLastOID	LastOID	Last OID processed by avAddRecord, avGetFeature, avReturnValue, avSetValue and avSetValueG
ugLastTheme	LastTheme	Last Theme processed by avAddRecord, avGetFeature, avReturnValue, avSetValue and avSetValueG
ugLayer	Layer	ILayer Object
ugLayerIndx	LayerIndex	Index into TOC for ugLayer
ugLayerStrg	LayerString	Name of ugLayer / ugTable
ugpFCls	FeatureClass	FeatureClass of ugLayer
ugPI	PI	PI value: 3.14159265358979
ugpProDesc	Description	Progress bar description
ugpProCancel	Cancel	Progress bar cancel status
ugpSR	SpatialReference	SpatialReference Object
ugsearchstring	SearchString	Search string
ugShapeT	ShapeT	IGeometry Object
ugSketch	Sketch	Sketch session flag
ugsnapTol	SnapTol	Snap Tolerance: 0.01
ugsnapTolMode	SnapTolMode	Snap Tolerance Mode (A/P)
ugTable	Table	IStandaloneTable Object
ugtheFTab	FTab	IFields Object
ugTolV1	TolV1	Tolerance Value: 0.005
ugTolV2	TolV2	Tolerance Value: 0.009
ugTolV3	TolV3	Tolerance Value: 0.00005
ugTolV4	TolV4	Tolerance Value: 0.00009
ugTolV9	TolV9	Tolerance Value: 0.0009
ugUpdateTOC	UpdateTOC	Automatic update of TOC
ugvDialogBoxData	VDialogData	Vertical Dialog contents
ugvDialogBoxLeft	VDialogLeft	Vertical Dialog X position
ugvDialogBoxRight	VDialogRight	Vertical Dialog Y position
ugvFileFrmLeft	FileDialogLeft	File Dialog X position

Public Variable Name	Property	Description
ugvFileFrmRight	FileDialogRight	File Dialog Y position
ugWinStyle	WinStyle	Window Style for avExecute
ugWrkSpcDesc	WrkSpcDesc	Workspace description for ugLayer
ugWrkSpcType	WrkSpcType	Workspace type for ugLayer
ugxPick	LastX	X coordinate of last pick
ugXYunits	XYunits	Spatial reference XY units
ugyPick	LastY	Y coordinate of last pick
